Representing Functional Data as Smooth Functions

A Master Thesis presented

by

Jan Ulbricht

(139316)

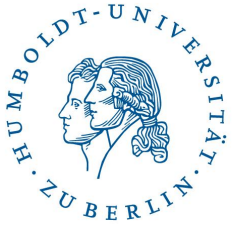
to

Prof. Dr. Wolfgang Härdle

CASE - Center of Applied Statistics and Economics

Institute of Statistics and Econometrics

Humboldt University, Berlin



in partial fulfillment of the requirements

for the degree of

Master of Science

Berlin, August 27, 2004

Declaration of Authorship

I hereby confirm that I have authored this master thesis independently and without use of others than the indicated resources. All passages, which are literally or in general matter taken out of publications or other resources, are marked as such.

Jan Ulbricht

Berlin, August 27, 2004

Acknowledgement

I would like to thank Prof. Dr. Wolfgang Härdle and also my second examiner Prof. Dr. Bernd Rönz for giving me the possibility to write this thesis.

I am especially indebted to Michal Benko for his excellent support all the time. We developed the main concept for implementing functional data analysis in XploRe, and the great number of long and fruitful discussions has given rise to a lot of new ideas. Furthermore, my profound thank is for Szymon Borak and his support in getting the dynamically linked library run.

Last but not least I would like to thank my friends for their encouragement and understanding all the ways during this study, and my parents for their generous financial support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis considers data which are functional in that each observation is a real function. Most often the functions are discretely sampled, that means there is only a finite set of distinct points with corresponding observed functional values which must be used to estimate the underlying function. The basis function approach reduces the estimation of a function to the estimation of a finite linear combination of known basis functions. Since dealing with functional data and its analysis is very computationally intensive, this thesis provides furthermore an implementation approach using the interactive statistical computing environment XploRe. Chapter 2 gives a short introduction to the nature of functional data, its representation by the basis function approach, and the main principles of implementation.

Chapter 3 deals in more detail with the theoretical background as well as aspects of implementation of the three most important families of basis functions: the Fourier basis, the polynomial basis and the B-spline basis. The main emphasis is put on the latter.

Chapter 4 describes different smoothing methods for estimating the coefficients of the basis expansions. Chapter 5 finally summarizes the main important facts and provides a short outlook on further aspects of functional data analysis and their implementation.

Chapter 2

Functional Data

2.1 Definition of Functional Data

In an increasing number of problems in a wide range of fields, the data observed are not the univariate or multivariate observations of classical statistics, but are functions attributable to an underlying infinite dimensional process.

Experimental economics, for instance, investigates the willingness to pay depending on the amount of risk, which is a natural continuum. Stock and option prices in finance are often treated as functions of time as in the Black-Scholes formula, and implied volatilities display time dependent functional patterns across strikes K and term structure τ, see Black & Scholes (1973). Growth curves, reaction time distributions, and learning curves evolve continuously over time. In engineering it is quite common to model continuous technical dynamic systems, and the EEG and EMG records in medicine depict continuous processes. All these examples share the property of being functionals of a continuous variable, most often of time. The sophisticated on-line sensing and monitoring equipment which is now routinely used for data collection in many fields makes it possible for the study of data which are functions to be done. But this changes the kind of the underlying sample space.

Formally, a given sample $\mathcal{X} = \{X_1, \ldots, X_n\}$ of independent observations is the measurable mapping $X : (\Omega, \mathfrak{A}, \mathrm{P}) \to (\mathcal{H}, \mathfrak{H})$, where $(\Omega, \mathfrak{A}, \mathrm{P})$ is a probability space, with Ω the set of all elementary events, a sigma algebra $\mathfrak{A}$ of subsets of

Ω, and a probability measure P defined on $\mathfrak{A}$; $\mathcal{H}$ is a measurable sample space and $\mathfrak{H}$ is a sigma algebra on $\mathcal{H}$. In the case of multivariate data $\mathcal{X}$ is a $(n \times p)$ random matrix containing n observations of p-dimensional row vectors of p (one-dimensional) random variables ($2 \leq p < \infty$), and $\mathcal{H}$ is a p-dimensional vector space, e.g. the p-dimensional Euclidean space $\mathbb{R}^p$. If $\mathcal{X}$ consists of random functions then $\mathcal{H}$ turns into a function space.

Since it is a function we denote the ith observation as $X_i(t), i = 1, \ldots, n$ of some argument t. It will be assumed that the functions are observed on a finite interval $J = [t_L, t_U] \subset \mathbb{R}$, where t_L and t_U denotes the lower and upper bound, respectively. In the case when t is interpreted as time the random function $X_i(t)$ is called the ith realization of a continuous stochastic process $X(t) \in \mathcal{H}$. Since this is mostly the case, we will refer to stochastic processes in the following.

In many cases, the observations will be single functions, but there are also applications where the functions are surfaces observed over two- or three-dimensional space, such as implied volatilities or spatial data. More generally, if $X_i(t)$ is a vector of m variables $X_{i1}(t), \ldots, X_{im}(t)$ then the observed data curves $\{X_i(t)\}_{i=1}^n$ are independent realizations of an m-dimensional continuous stochastic process.

When analyzing sampled random functions, it is natural to consider each curve as a single observations, to summarize the functions in terms of a mean function, and to measure in some way the variation of the functions about this mean. Ramsay & Dalzell (1991) introduce the name functional data analysis (FDA) to the analysis of data of this kind which leads to the following definition:

Definition 1 (Functional Data) *A sample $\mathcal{X} = \{X_1, \ldots, X_n\}$ is called functional data when the ith observation is a real function $X_i(t), t \in J, i = 1, \ldots, n$, and hence, each $X_i(t)$ is a point in some function space $\mathcal{H}$.*

To avoid confusion, a single functional datum, that means a single observed function, is a called a replication. Functional data in turn is a random sample of replications. If the argument t is interpreted as time, the ith replication is also referred to as ith realization of the underlying continuous stochastic process $X(t)$.

According to Ramsay (1982) there were two lines of development in dealing with functional data. The first had been the expression of traditional data analytic technology in the language of functional analysis. This goes back in particular to the French school of analyse des données, see the monographs of Cailliez & Pagès (1976) and Dauxois & Pousse (1976) for introduction. The second line of development had been the statistical application of spline functions, especially in the scope of nonparametric function estimation. See Silverman (1985), and Green & Silverman (1994) for details. In recent years, an increasing number of publications on functional data analysis has been evolved. Among them were applications in various fields of science as well as theoretical essays. With Ramsay & Silverman (1997) and Ramsay & Silverman (2002) also two textbooks on functional data analysis are available.

Many FDA methods are adaptions of classical multivariate methods such as principal components analysis (PCA), linear modeling, and analysis of variance. When analyzing the variation within and between functions, derivatives will be used extensively. Furthermore, the related concepts of roughness and smoothness of functions play a crucial role. These concepts will be introduced in the next section.

2.2 Roughness and Smoothness of Functions

Due to the insight of Ramsay (1997), it is the smoothness of the process generating functional data that differentiates this type of data from more classical multivariate observations. This smoothness ensures that the information in the derivatives of functions can be used in a reasonable way.

Derivatives also play an important role in scientific models. Often it is more interesting how rapidly and to which amount a system responds rather than its actual level of response. Furthermore, as described by Ramsay & Silverman (2002) the first and the second derivatives can reflect the energy exchange within a system. Especially in the natural sciences and in engineering, derivatives are also needed to

construct models for data based on differential equations. In many fields such as pharmacokinetics and industrial process control differential equations are especially useful when feedback or input/output models are to be developed to control the behavior of systems.

For ease of notation for $m = -1, 0, 1, 2, \ldots$ we introduce the differential operator

$$D^m f(t) \stackrel{\text{def}}{=} \frac{d^m f(t)}{dt^m} = f^{(m)}(t), m \geq 1, \tag{2.1}$$

with $D^0 f(t) = f(t)$ and

$$D^{-1} f(t) = \int_{t_L}^{t} f(s) ds.$$

Furthermore, the concept of inner products is needed. In the scope of this paper it is sufficient to define inner products only for real function spaces. For a more general definition, see, for example, Simmons (1963).

Definition 2 *An inner product on the real function space $\mathcal{H}$ is a function $\langle \cdot, \cdot \rangle$ defined on $\mathcal{H} \times \mathcal{H}$ with values in $\mathbb{R}$ and satisfying the properties*

1. $\langle ax + y, z \rangle = a\langle x, z \rangle + \langle y, z \rangle$,
2. $\langle x, y \rangle = \langle y, x \rangle$,
3. $\langle x, x \rangle \geq 0$ *and* $\langle x, x \rangle = 0$, *iff* $x = 0$

for all $x, y, z \in \mathcal{H}; a, b \in \mathbb{R}$.

One easy way to quantify the degree of smoothness of a function is to use the adjoined concept of roughness. Following Leurgans, Moyeed & Silverman (1993), roughness is a measure of the rapidity of variability of a function $f(t)$. It is generated by a positive definite quadratic form defined for functions f, g with square integrable mth derivative as

$$PEN_m(f, g) \stackrel{\text{def}}{=} \int_J D^m f(t) D^m g(t) dt. \tag{2.2}$$

Here, the notation $PEN_m(\bullet)$ is used because the concept of roughness will be primarily needed as penalty terms for smoothing techniques. See Chapter 4 for details. It is obvious that the roughness is an inner product.

The roughness of any particular function f is then quantified by setting $f = g$ in the quadratic form, to give

$$PEN_m(f) = \int_J \{D^m f(t)\}^2 dt. \tag{2.3}$$

This allows to measure the closeness of $f(t)$ to a polynomial of order m, see Definition 5.

One special case of roughness is total curvature. The curvature of a function $f(t), t \in J$ is defined as its squared second derivative $\{D^2 f(t)\}^2$ and measures the deviation of $f(t)$ from being a linear function at t. Integrating on all arguments, respectively the whole interval of interest J, yields the total curvature as

$$PEN_2(f) = \int_J \{D^2 f(t)\}^2 dt. \tag{2.4}$$

Often there is a need for a wider class of measures of deviation. Especially, when there is periodicity in the data or an exponential trend, it would not be sufficient to use the integrated squared mth derivative, because it can only penalize deviations from polynomials. On the other hand, it may be that, locally at least, the function $f(t)$ should ideally satisfy a particular differential equation, and we may wish to penalize departure from this. This can be done by using a linear differential operator defined as follows:

Definition 3 *The linear differential operator (LDO) L is defined as*

$$Lf(t) = w_0(t)f(t) + w_1(t)Df(t) + \ldots + w_{m-1}(t)D^{m-1}f(t) + D^m f(t), \tag{2.5}$$

where the coefficients $w_i, i = 0, \ldots, m-1$ may either be constants or functions of $t \in J$.

The roughness is then given by $PEN_L(f) = \int_J \{Lf(t)\}^2 dt$. It is obvious that (2.3) and (2.4) are special cases of LDOs.

Depending on the underlying LDO the degree of smoothness of a function $f(t), t \in J$ can be derived from the numerical value of the accompanied roughness. Since roughness is a quadratic form, the degree of smoothness is also non-

negative. The lower the roughness, the higher is the smoothness of the function. If $PEN_L(f) \equiv 0$ then f is called to be "hypersmooth".

The concept of smoothness is essential in Chapter 4 when the coefficients of the basis expansions will be estimated. The degree of smoothness is one competing factor in smoothing discretely observed functional data by functions, besides the squared bias.

2.3 Observed Functional Data

Although in functional data analysis the ith observation is a real function, functional data are usually observed and recorded discretely. Typically, the sample data contain a number of n independent replications and the record of replication $X_i(t), i = 1, \ldots, n$ might consist of n_i pairs $\{t_{ij}, y_{ij}\}, j = 1, \ldots, n_i$, where t_{ij} denotes the argument, and y_{ij} the observed functional value.

The choice of t_{ij} is very nonrestrictive, e. g. the argument values may vary between the records and need not to be equally spaced. Furthermore, the number of observations n_i can differ between the records. But nevertheless, the argument should lie in the range of values of interest, that means $t_{ij} \in J$ for all i, j.

Usually the functional observations could not be observed without some random noise. Let $\mathbf{y_i} = (y_{i1}, \ldots, y_{in_i})^\top$, $\mathbf{t_i} = (t_{i1}, \ldots, t_{in_i})^\top$, and $\epsilon_\mathbf{i} = (\epsilon_{i1}, \ldots, \epsilon_{in_i})^\top$. An adequate model of the underlying relationship between sample data and the true functional values $X_i(\mathbf{t_i})$ is

$$\mathbf{y_i} = X_i(\mathbf{t_i}) + \epsilon_\mathbf{i}, \tag{2.6}$$

where $\epsilon_\mathbf{i}$ is assumed to be an unobservable error term with $\epsilon_\mathbf{i} \sim \mathrm{N}(0, \Sigma_i)$ and $\Sigma_i = \mathrm{diag}(\sigma^2_{i1}, \ldots, \sigma^2_{in_i})$. The observed data $\mathbf{y_i}$ are separated into two components, where the structural component $X_i(\mathbf{t_i})$ has some simple or interpretable or otherwise interesting low dimensional structure and the residual component $\epsilon_\mathbf{i}$ is viewed as noise, unwanted variation or possibly as data ready for further exploration. This separation is possible because we observe several realizations of the underlying stochastic process.

In a first step these discrete sample data must be put into functional form, or more precisely into smooth functions. Since some random noise is included in the data the appropriate class of methods are smoothing techniques.

2.4 The Basis Function Approach

For representing functional data as smooth functions a flexible method is needed that can track local curvature and provides a sufficient number of derivatives. One such method is the basis function approach, a generalization of the linear regression approach. The main idea is to replace the vector of inputs t with transformations of t, and then to use linear models in this new space of derived input features, see Hastie, Tibshirani & Friedman (2001).

In order to develop the theory of the basis function approach, some assumptions of the underlying stochastic process $X(t), t \in J$ are necessary. In the following the space $\mathcal{H}$ is assumed to be a separable Hilbert space of measurable functions. The first and second order moments of $X(t)$ are assumed to exist and to be finite:

$$X_\mu(t) = \mathrm{E}\, X(t), t \in J, \tag{2.7}$$

$$\mathrm{Var}_X(t) = \mathrm{E}\{X(t) - \mathrm{E}\, X(t)\}^2, t \in J, \tag{2.8}$$

$$\Gamma_X(s,t) = \mathrm{E}\{X(s) - \mathrm{E}\, X(s)\}\{X(t) - \mathrm{E}\, X(t)\}, s,t \in J. \tag{2.9}$$

In order to simplify the notation, it is assumed that $X_\mu(t) = 0, t \in J$.

Definition 4 *A real function $K(s,t)$ with the properties*

1. $K(s,t) = K(t,s)$,
2. $\sum_{s=1}^{N}\sum_{t=1}^{N} K(s,t)\alpha_s\alpha_t \geq 0$, *for any* $N \geq 1; \alpha_s, \alpha_t \in \mathbb{R}; s,t \in J$

is called a positive-definite kernel on J^2.

The corresponding space of functions $\mathcal{H}_K$ is called a reproducing kernel Hilbert space (RKHS), since $K(s,t)$ possesses the reproducing property of $\mathcal{H}_K$

$$\langle K(\cdot,s), K(\cdot,t)\rangle_{\mathcal{H}_K} = K(s,t), \tag{2.10}$$

where $\langle \cdot, \cdot \rangle_{\mathcal{H}_K}$ denotes the inner product induced by K. It is obvious that $\Gamma_X(s,t)$ is a kernel. Hence, for every continuous stochastic process with finite second order moments there exists a RKHS. See e.g. Istrăţescu (1987) for the proof and a further discussion of reproducing kernels.

The concept of RKHS is used to estimate the ith realization of a continuous stochastic process out of a finite set of discrete observations $\{t_{ij}, y_{ij}\}_{j=1}^{n_i}$. Equation (2.6) suggests a regression approach. One possibility is to minimize the penalized residual sum of squares

$$\min_{X_i \in \mathcal{H}_K} \left[\{\mathbf{y_i} - X_i(\mathbf{t_i})\}^\top \{\mathbf{y_i} - X_i(\mathbf{t_i})\} + \lambda PEN_2(X_i) \right], \tag{2.11}$$

where λ is a smoothing parameter. More details on penalized regression are given in Section 4.1. Wahba (1990) shows that the solution to (2.11) and even to a more general class of penalized regression problems has the form of the finite-dimensional linear combination

$$\hat{X}_i(t) = \sum_{k=1}^{K} c_{ik}\phi_k(t), c_{ik} \in \mathbb{R}; K < \infty \tag{2.12}$$

where $\phi_k(t) = K(k,t)$ is a real-valued function whose values at $t \in J$ is $K(k,t)$. The function $\phi_k(t)$ is known as the representer of evaluation at k in $\mathcal{H}_K$, since for $X_i(t) \in \mathcal{H}_K$, it holds that $\langle K(\cdot,k), X_i \rangle = X_i(k)$, see Hastie, Tibshirani & Friedman (2001).

It can be shown that $\hat{X}_i(t)$ in (2.12) converges to $X_i(t)$ in quadratic mean as $K \to \infty$ uniformly in $t \in J$. See Wahba (1990) for the proof. It will be assumed that at least part of the variation of the ith replication can be accounted for in terms of (2.12). Hence, to ease notation we write $X_i(t)$ instead of $\hat{X}_i(t)$.

Following Hastie, Tibshirani & Friedman (2001) the functions $\phi_k(t)$ are referred to as basis functions, since they are basis functions of a K-dimensional subspace of $\mathcal{H}_K$. For details on the so called "partitioning principle" see Ramsay & Dalzell (1991). Hence, (2.12) is a basis function expansions, or shorter basis expansion. The estimation of $X_i(t)$ using (2.12) is therefore called the "Basis Function Approach".

In order to ease notation the following vectors and matrices are introduced introduced for a finite dimensional vector $\mathbf{t_i}$ of arguments

$$X_i(\mathbf{t_i}) = \Phi(\mathbf{t_i})\mathbf{c_i}, \tag{2.13}$$

where $X_i(\mathbf{t_i}) = \{X_i(t_{i1}), \ldots, X_i(t_{in_i})\}^\top$,

$$\Phi(\mathbf{t_i}) = \{\phi_1(\mathbf{t_i}), \ldots, \phi_K(\mathbf{t_i})\} = \begin{pmatrix} \phi_1(t_{i1}) & \ldots & \phi_K(t_{i1}) \\ \vdots & \ddots & \vdots \\ \phi_1(t_{in_i}) & \ldots & \phi_K(t_{in_i}) \end{pmatrix}$$

is a $(n_i \times K)$ matrix and $\mathbf{c_i} = (c_{i1}, \ldots, c_{iK})^\top$.

The basis function approach impose a finite-dimensional structure on the estimation problem, by restricting the choice of $X_i(t)$ to the span of a prescribed set of basis functions $\phi_1, \ldots, \phi_K$. Estimating the ith realization $X_i(t)$ of a continuous stochastic process reduces to the use of a suitable set of simple structured basis functions and the estimation of the unknown coefficients c_{ik} of the basis expansion.

In the scope of functional data analysis the same basis is used for all replications, since an adjusting to the requirements of one specific replication $X_i(t)$ can sufficiently be made by the vector of coefficients $\mathbf{c_i}$. In the case of multiple functions, the basis expansion might also vary between the variables, as in the case of simultaneously involved periodic and non-periodic variables.

The advantage of the basis function approach is that nonlinearity and local features in the data can be easily absorbed by the basis functions while the model is still linear in the transformations. Also, when computing the mth derivative of $X_i(t)$, this simplifies to computing the mth derivative of the basis functions. Hence, the property of being smooth is primarily left to appropriate basis functions. When evaluating a sample of functional observations computational effort is reduced, because the basis functions have only to be calculated once.

The more basis functions are involved, the more complex the fitted function can be. The degree to which the data $\mathbf{y_j}$ are smoothed, rather than exactly reproduced or interpolated, is determined by the number K of basis functions, see Ramsay (1997).

Several basis function approaches will be presented in Chapter 3. On how to estimate the coefficients $\mathbf{c_i}, i = 1, \ldots, n$ will be dealt with in Chapter 4.

2.5 Implementing Functional Data Analysis in XploRe

Since dealing with functional data and its analysis is very computationally intensive, some suitable software is necessary. The main problem arises from the fact that functional data are functions but usually it is not possible to store a functional mapping on a computer directly. Often, we can only store discrete arrays.

The interactive statistical computing environment XploRe is a matrix based language which allows for the computation of arrays up to eight dimensions. The main advantage for FDA purposes is that XploRe allows to store a functional mapping directly when it is already known before runtime by a `proc-endp` environment. This makes it especially easy to design LDOs with variable coefficients, as will be illustrated later on. Furthermore, XploRe has the possibility to include dynamically linked libraries (DLLs). This allows for fast computation of complex algorithms by using other softwares, such as C++. How to store functional data for further analysis will be described in the next subsection.

2.5.1 Objects for Functional Data Analysis in XploRe

For a single functional replication several pieces of information are needed at least to identify one specific functional basis, e.g. the number and type of basis functions, parameters for the basis functions, or the range of interest, see Ramsay (2003). Fortunately, XploRe has the capacity to define a single compound variable that can contain several pieces of information of varying types, such as scalars, vectors, strings, arrays etc. These are list variables.

In the spirit of object-oriented programming a class is a specification for a list that pre-specifies its structure. According to Ramsay (2003) a class specifies the type and gives a name for each piece of information in the compound variable but furthermore, gives a type name to the compound object itself. Assigning a class

name to a list allows the XploRe interpreter to know in advance what its internal elements are.

Objects are specific variables created so as to conform to the specification in a specific class. In the functional data context, for example, a functional data object, called an fd object, in its simplest version conforms to the functional data class that specifies that the object will have an array of numbers that are the coefficients for basis function expansion, and another object called fdbasis object that specifies a basis for the expansion, as described in Ramsay (2003).

There are four different objects needed for FDA in XploRe:

- an fdbasis object that specifies a basis for the expansion,
- an fd object that contains an array of coefficients for the basis expansion and the underlying fdbasis object,
- a linear differential operator called LDO object, what in turn is only necessary when dealing with LDOs with variable coefficients, and
- a bifd object which consists of an array of coefficients and two underlying fdbasis objects. They arise naturally when dealing with covariance and correlation functions of fd objects. bifd objects are different in the way that they depend on two arguments.

fdbasis objects are created by the quantlet `createfdbasis`. They are a list consisting of the elements "`type`", "`range`", "`nbasis`", and "`params`". See Section 3.5 for further details. fd objects are created by the quantlet `data2fd`. They are a list consisting of the elements `coef`, and the underlying fdbasis object. The quantlet `data2fd` will be described in Section 4.3. Generating LDO objects contains of two parts. First, the user has to define the LDO in a `proc-endp` environment. Afterwards the LDO object must be created, either by using the `createLDO` quantlet or by directly using the list concept. The following example illustrates the implementation of the LDO $L = w_1 D + w_2 D^3$ with $w_1(t) = 1.25t$ and $w_2(t) = 0.5t^2$:

```
proc (wt) = LDO1 (evalarg)
wt = matrix (2, rows (evalarg), cols (evalarg))
wt[1] = 1.25 .* evalarg
wt[2] = 0.5 .* evalarg^2
endp
LDO = createLDO ("LDO1", #(1, 3))
```

LDOexample.xpl

The input parameter is the matrix of argument values, the output parameter is an array where the first dimension coincides with the number of summands in the LDO, the second and third are the rows and columns of the input parameter, respectively. Note that this form must always be held. The single summands of the LDO (here `wt[1]` and `wt[2]`) will be defined and computed inside the environment. Afterwards the LDO object can be created using the quantlet createLDO. Its first input parameter is a string containing the name of the proc-endp environment where the LDO is defined, the second is a vector containing the number of derivatives which will be needed for the single summands of the LDO. The output is an LDO object, here called "LDO".

Up to now bifd objects are only needed for specific transformations of basis expansions as in the case of covariances. From the theoretical point of view, there could also be an underlying stochastic process depending on more than one argument. In practice, it might be difficult to find another continuum that is observed additionally to time. Therefore in most cases it is sufficient to deal with several variables but with the same argument.

2.5.2 Three Steps in an FDA

Figure 2.1 illustrates the three typical steps in doing functional data analysis. In a first step, the fd object has to be created out of the discrete input data. According to the basis function approach in Section 2.4 this can be divided into two sub-steps.

At the beginning, a system of basis functions must be specified. This will contain different types of information, such as a string indicating the type of basis,

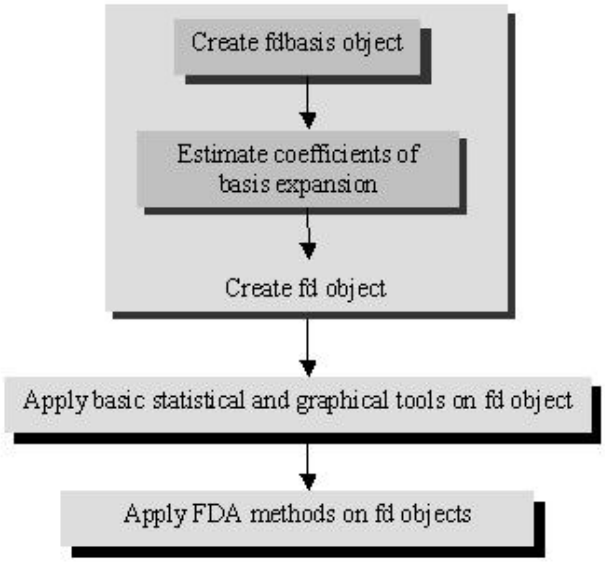


Figure 2.1: Three Steps in Functional Data Analysis.

the lower and upper bounds of the interval of interest as a vector containing two elements, the number of basis functions as an integer, and an array of parameters which specify the basis. All this is stored in a functional object called fdbasis. How to create and evaluate fdbasis objects will be described in Section 3.5. Afterwards, the fdbasis object will be used to estimate to coefficients of the basis expansion. Smoothing Methods for functional data are described in Chapter 4. Hence, the fd object will contain two elements, an array of coefficients for the sample of functional observations and the fdbasis object it refers to. How to create and evaluate fd objects will be shown in Section 4.3.

In a second step, basic statistical and graphical tools will be applied to the fd object. This covers functions used to display and summarize functional data, such as means, standard deviations, variances, covariances and correlations. Also one initial step in functional data analysis, a method called registration, might be placed here. The third step finally applies the typical FDA methods to the fd object. This paper only covers the first step.

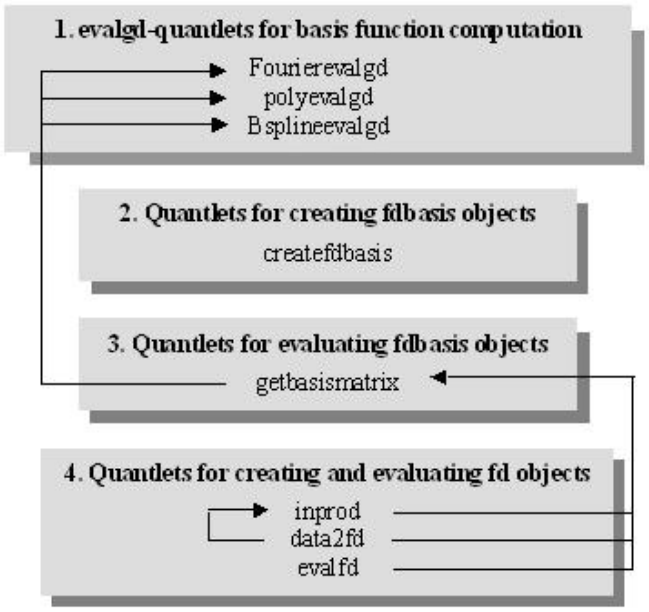


Figure 2.2: System of Quantlets for FDA.

Efficient implementation requires a system of quantlets which is shown in Figure 2.2, where an arrow indicates the direct use of a quantlet at a lower stage or a lower position, respectively.

The system architecture consists of hierarchically ordered phases. This enables a simple handling of future extensions. In general, only the quantlets at one stage must be adjusted then. For example, when implementing localized basis function estimation (see Section 4.4 for details) only the quantlets `data2fd` and `evalfd` at the fourth stage must be adjusted. Of course, the already implemented quantlets just cover the most important cases. Therefore remarks and hints on future extensions are given in the following chapters as often as possible.

The computation of basis functions, their derivatives as well as linear differential operators applied on them is essential for functional data analysis. The main important basis expansions and their computation are presented in the next chapter.

Chapter 3

Basis Expansions

The basis functions will be chosen so that their span includes good approximations to most smooth functions. Since they have to represent the underlying structure in the sample data they must be able to flexibly exhibit the required curvature where needed, but also to be nearly linear when appropriate. Furthermore, for computational reasons they should be fast to compute and easy to evaluate. Especially for FDA purpose they must be differentiable as often as required.

This paper covers the three most common bases: Fourier bases, polynomial bases, and B-spline bases. Fourier series are known to model periodic functions and they also might be useful for periodic data with a fixed and known frequency. They are dealt with in Section 3.2. Polynomials are easy to compute but also very inflexible. They seem to be appropriate only for simple data structures without a lot of local features. Polynomials will be introduced in Section 3.3. B-splines are a very often used class of bases for non-periodic data in the functional data context. Especially, their compact support and the fast computation as well as the ability to create appropriate and smooth approximations of the underlying data favor their extensive usage. They will be treated in Section 3.4. Finally Section 3.5 collects all necessary information in order to compute fdbasis objects in XploRe.

Ramsay (2003) provides additional basis expansions, such as the exponential basis with

$$\phi_k(t) = \exp\{\alpha_k t\}, k = 1, \ldots, K, \tag{3.1}$$

where α_k is a rate parameter, and the power basis, consisting of a sequence of possibly non-integer powers, including negative powers, of the argument t that is required to be positive. Another possible class of basis functions arises out of expanding a square integrable function in terms of wavelet series. The discrete wavelet transform (DWT) is a fast, orthogonal series expansion that operates on a data vector whose length is an integer power of two, transforming it into a numerically different vector of the same length. DWT can be viewed as a rotation in function space, from the input space or time domain, where the basis functions are the unit vectors, to a different domain, see Press et al. (2002). For an introduction to wavelets see e.g. Härdle et al. (1998), or Daubechies (1992). XploRe already provides the `twave` library to deal with wavelets. Nevertheless using wavelet bases is beyond the scope of this paper.

For users with special bases in mind that are not available so far the next section gives an overview on the general concept of implementing basis expansions in XploRe.

3.1 Basis Expansions in XploRe

Basis expansions in XploRe are provided by the so called evalgd quantlets. They contain all specific computation algorithms for a certain class of basis expansions. Currently, the existing evalgd quantlets

- `Fourierevalgd`,
- `polyevalgd`, and
- `Bsplineevalgd`

allow for computing the actual basis and all of its derivatives. If, for example, for future purpose the definite integrals of a basis are needed, then the necessary algorithms might also to be included in these quantlets.

The evalgd quantlets can only compute derivatives, but not LDOs. The quantlet `getbasismatrix` at the next stage (see Figure 2.2) will be needed for that

purpose. The idea behind this decomposition is to include the check, whether the corresponding input parameter is an LDO object, only once in the `getbasismatrix` quantlet, instead several times in the evalgd quantlets. This reduces extension expenditure. Furthermore, if some additional operator checks, such as for nonlinear differential operators, may be needed in future, they should also be included in the `getbasismatrix` quantlet.

The evalgd quantlets are needed to have the following input parameters, maybe in a different order:

argvals: a $(n \times p)$ matrix of argument values which the bases should be evaluated at. To use a matrix will be primarily necessary when evaluating bifd objects, or during the estimation of the basis expansion coefficients if several replications are observed for different arguments.

deriv: a $(r \times 1)$ vector of non-negative integers, indicating the order of derivatives to compute. The possibility to compute several derivatives simultaneously is especially necessary for using LDOs because computation time is reduced.

params: some additional parameters, needed for identification or specification purposes. E.g. the period when using Fourier bases (see Section 3.2).

The output is a $(n \times K \times r \times p)$ array, where the first dimension n corresponds with the number of rows in argvals, the second dimension K corresponds to the number of basis functions, the third dimension r corresponds to the number of derivatives to compute, and the fourth dimension p corresponds to the number of columns in argvals.

In the following three section, the already implemented evalgd quantlets and their theoretical background will be displayed.

3.2 Fourier Bases

```
phi = Fourierevalgd (xvec, nbasis, period{, deriv})
      evaluates the fourier basis on xvec
```

Trigonometric functions are most widely used for approximating periodic functions. As known from theory of Fourier series a periodic function $f(t)$ can be represented in terms of a finite or infinite Fourier series

$$\hat{f}(t) = c_0 + c_1 \sin \omega t + c_2 \cos \omega t + c_3 \sin 2\omega t + c_4 \cos 2\omega t + \dots. \tag{3.2}$$

where $\omega = \frac{2\pi}{T}$ is a known frequency. The period T usually coincides with the interval J of interest. For their additive structure Fourier series can be easily used as basis expansion.

Let K be an even integer, then the Fourier bases is defined by

$$\phi_0(t) = \frac{1}{\sqrt{T}}, \tag{3.3}$$

$$\phi_{2r-1}(t) = \frac{1}{\sqrt{T/2}} \sin r\omega t, \tag{3.4}$$

$$\phi_{2r}(t) = \frac{1}{\sqrt{T/2}} \cos r\omega t, \tag{3.5}$$

for $r = 1, \dots, K/2$. By the coefficients $\frac{1}{\sqrt{T}}$ and $\frac{1}{\sqrt{T/2}}$, the basis functions become orthonormal, that means

$$\int_T \phi_{k_1}(t)\phi_{k_2}(t)dt = \begin{cases} 0, & k_1 \neq k_2, \\ 1, & k_1 = k_2 \end{cases} \tag{3.6}$$

for $k_1, k_2 = 0, \dots, K$.

Since each derivative of a trigonometric function is a shift by $T/4$ to the right, the mth derivatives of Fourier bases can be computed by

$$D^m\phi_0(t) = 0, m \geq 1 \tag{3.7}$$

$$D^m\phi_{2r-1}(t) = (r\omega)^m \sin(r\omega t + \frac{m\pi}{2\omega}), m \geq 0, \tag{3.8}$$

$$D^m\phi_{2r}(t) = (r\omega)^m \cos(r\omega t + \frac{m\pi}{2\omega}), m \geq 0 \tag{3.9}$$

for $r = 1, \dots, K/2$.

The quantlet `Fourierevalgd` allows to compute Fourier bases and one or more of their derivatives simultaneously for a matrix of arguments. Its syntax is

```
phi = Fourierevalgd (xvec, nbasis, period{, deriv})
```

with input parameters

`xvec` - a $(p \times q)$ matrix of arguments where the Fourier bases should be evaluated at.

`nbasis` - a scalar, dimension of the functional basis

`period` - a scalar, period of functions

`deriv` - a $(r \times 1)$ vector containing order of derivative(s), if not the actual bases will be computed

This quantlet returns a $(p \times$ `nbasis` $\times r \times q)$ array, where one row contains the evaluated bases for one argument. One example of actual Fourier bases for period $T = 10$ and $K = 5$ is given in Figure 3.1.

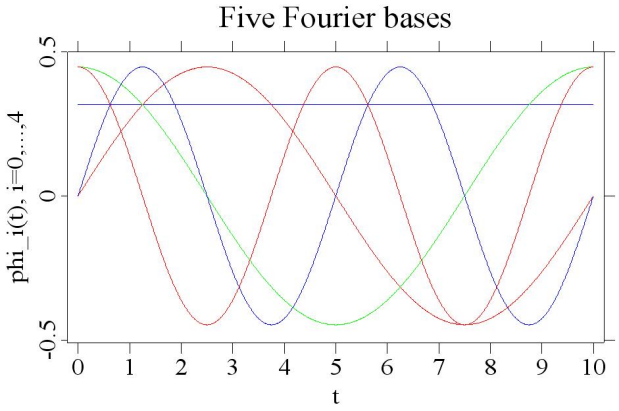


Figure 3.1: Five Fourier Bases for period $T = 10$.

`fb1.xpl`

A Fourier series is especially useful for extremely stable functions, meaning functions where there are no strong local features and where the curvature tends

to be of the same order in the whole interval J. They generally yield expansions which are uniformly smooth, but they are inappropriate to some degree for data known or suspected to reflect discontinuities in the function itself or in low-order derivatives. Also, they cannot exhibit very local features without using a large number of bases. See Ramsay & Silverman (1997).

3.3 Polynomial Bases

```
phi = polyevalgd (xvec, nbasis{, deriv{, omega}})
      computes polynomial basis matrix
```

Besides trigonometric functions, which are appropriate to approximate periodic functional behavior, polynomials are used for approximating non-periodic functions. According to the notation of de Boor (1978) and Schumaker (1981), we introduce the following definition.

Definition 5 (Polynomial) *A polynomial of order K is defined as*

$$p(t) = c_0 + c_1 t + \ldots + c_{K-1} t^{K-1} = \sum_{k=0}^{K-1} c_k t^k. \tag{3.10}$$

This kind of definition will moreover be useful when dealing with B-splines in Section 3.4.

A slightly modified version of (3.10) can be used for basis function expansion. In this context, the k-th basis function is described as

$$\phi_k(t) = (t - \omega)^k, \quad k = 0, 1, \ldots, K-1. \tag{3.11}$$

In (3.11), ω represents a constant shift parameter, which helps to keep the polynomials from getting too ill-conditioned. The mth derivative of $\phi_k(t)$ is

$$D^m \phi_k(t) = k(k-1)\ldots(k-m+1)(t-\omega)^{(k-m)}, \quad k = 0, 1, \ldots, K-1, \tag{3.12}$$

for $m = 1, 2, \ldots$.

The quantlet `polyevalgd` computes polynomial bases and one or more of their derivatives simultaneously for a vector of arguments. Its syntax is

```
phi = polyevalgd (xvec, nbasis{, deriv{, omega}})
```

with input parameters

`xvec` - a $(p \times q)$ matrix of arguments where the polynomial bases should be evaluated at.

`nbasis` - a scalar, indicating the dimension of the functional basis

`deriv` - a $(r \times 1)$ vector containing order of derivative(s), if not the actual bases will be computed

`omega` - an optional scalar, indicating the shift parameter, if not omega will be set to zero

This quantlet returns a $(p \times$ `nbasis` $\times r \times q)$ array, where one row contains the evaluated bases for one argument. One example of actual polynomial bases for the interval $J = [0, 1.1]$ is given in Figure 3.2.

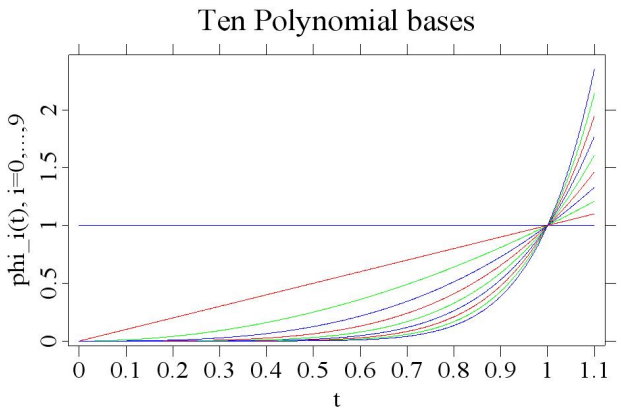


Figure 3.2: Ten polynomial bases for $J = [0, 1.1]$.

`pb1.xpl`

While polynomials are evaluated, differentiated and integrated easily and in finitely many steps using just the basic arithmetic operations, they also have some unpleasant limitations. With increasing order there arise strong oscillations. They are inflexible to capture local features and often only bad approximate at the end of large intervals. Furthermore, they are very sensitive to the choice of interpolation points and depend globally on local properties.

Nevertheless, polynomials seem to do well on sufficiently small intervals and with low order. This suggests that in order to achieve a class of approximating functions with greater flexibility, the interval of interest should be subdivided into smaller pieces, and the function might be approximated by locally defined polynomials of relatively low degree. This is the motivation behind piecewise polynomials, splines and finally B-splines, which are all dealt with in the next section.

3.4 B-spline Bases

B-splines are widely used for basis expansions in the FDA context. They can model sharp changes in the underlying function as well as its smooth variation. Their band-structured matrix of values, as well as a fast computation favor an extensive usage. Technically, B-splines are a basis of a certain subspace of the space of piecewise polynomials and therefore a wider scope of theory is necessary to introduce them. The concept of piecewise polynomial functions in Subsection 3.4.1, followed by the definition of polynomial splines in Subsection 3.4.2 provide the mathematical background of B-splines and their main properties, which will be dealt with in Subsections 3.4.3 to 3.4.5. Since the effort for computing B-splines is more extensive Subsection 3.4.6 is reserved for that.

3.4.1 Piecewise Polynomial Functions

Divide the interval $J = [t_L, t_U]$ into m subintervals by a strictly increasing sequence of points $\xi = \{\xi_i\}_{i=1}^{m+1}$, with $t_L = \xi_1 < \xi_2 < \ldots < \xi_m < \xi_{m+1} = t_U$. The points ξ_i are called breakpoints. The idea of piecewise polynomial functions is stated in the

following definition.

Definition 6 *Let K be a positive integer. Then the corresponding* ***piecewise polynomial function*** *$f(t)$ of order K is defined by*

$$f(t) \stackrel{\text{def}}{=} p_i(t) = \sum_{j=0}^{K-1} c_{ij} t^j I\{t \in [\xi_i, \xi_{i+1})\}, \quad i = 1, \ldots, m, \tag{3.13}$$

where

$$I(t \in A) \stackrel{\text{def}}{=} \begin{cases} 1, & t \in A \\ 0, & \text{otherwise} \end{cases}$$

is the indicator function defined on a set A.

Here the indicator function $I(\cdot)$ is used to make sure that only one piecewise polynomial is defined on each subinterval.

One convenient basis function representation of (3.13) is

$$f(t) = \sum_{i=1}^{m} \sum_{j=0}^{K-1} c_{ij} \phi_{ij}(t), \tag{3.14}$$

where

$$\phi_{ij}(t) = t^j I\{t \in [\xi_i, \xi_{i+1})\}. \tag{3.15}$$

Two piecewise polynomial functions agree if they consist of the same polynomial pieces, broken at the same points. The collection of all piecewise polynomial functions of order K with breakpoint sequence ξ is denoted by $\mathcal{P}_{K,\xi}$, which is clearly a linear space of dimension Km.

The main disadvantage of the above definition of piecewise polynomial functions is illustrated in Figure 3.3. Piecewise polynomial functions need not to be continuous or even smooth at the interior breakpoints.

This can be overcome by imposing the constraints

$$p_{i-1}(\xi_i) = p_i(\xi_i), \quad i = 2, \ldots, m, \tag{3.16}$$

to achieve a continuous piecewise polynomial function, and equivalently by

$$D^k p_{i-1}(\xi_i) = D^k p_i(\xi_i), \quad i = 2, \ldots, m, \quad k = 1, \ldots, K-2, \tag{3.17}$$

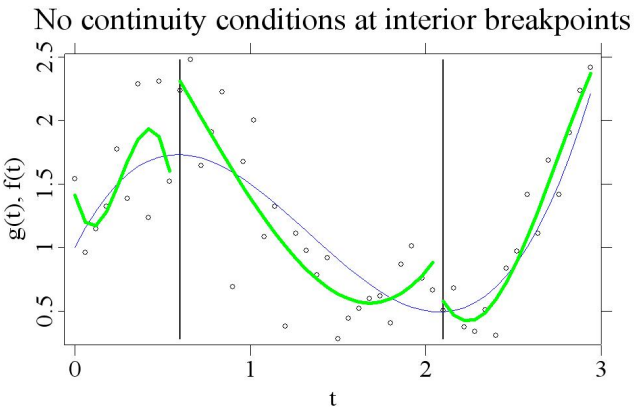


Figure 3.3: Piecewise Cubic Polynomials (thick) with $\xi = \{0, 0.6, 2.1, 3\}$ for $g(t) = 0.75t^3 - 3t^2 + 2.75t + 1 + \epsilon$ (thin), where $\epsilon \sim \mathrm{N}(0, 1/9)$.

pcp1.xpl

in order to obtain a continuous kth derivative of $f(t)$, and therefore an appropriate amount of smoothness at each interior breakpoint ξ_i. In other words, (3.16) and (3.17) are continuity conditions for $f(t)$ and its derivatives, respectively. Figures 3.4 - 3.6 illustrate different numbers of continuity conditions and their influence on the piecewise polynomial function.

Let the number of continuity conditions at each interior breakpoint $\xi_i, i = 2, \ldots, m$ be stored in the vector $\nu = \{\nu_i\}_{i=2}^m$, where $\nu_i = k$ means that $f(t)$ and all derivatives up to $D^{k-1}f(t)$ are continuous at ξ_i. As the conditions (3.16) and (3.17) are linear and homogenous, the subset of all $f(t) \in \mathcal{P}_{K,\xi}$ satisfying (3.16) and (3.17) for a given vector ν is a linear subspace of $\mathcal{P}_{K,\xi}$ and is denoted by $\mathcal{P}_{K,\xi,\nu}$. The continuity conditions lower the dimension of $\mathcal{P}_{K,\xi,\nu}$ to $Km - \sum_{i=2}^m \nu_i$ and therefore also the number of necessary basis functions in (3.14).

Unfortunately, the basis functions (3.15) are no longer appropriate for the basis expansion (3.14). One has to use the classical Lagrangian procedure or to eliminate unnecessary coefficients by solving a linear equation system. A more direct way in this case is to express (3.15) in terms of a suitable basis of $\mathcal{P}_{K,\xi,\nu}$, such as the

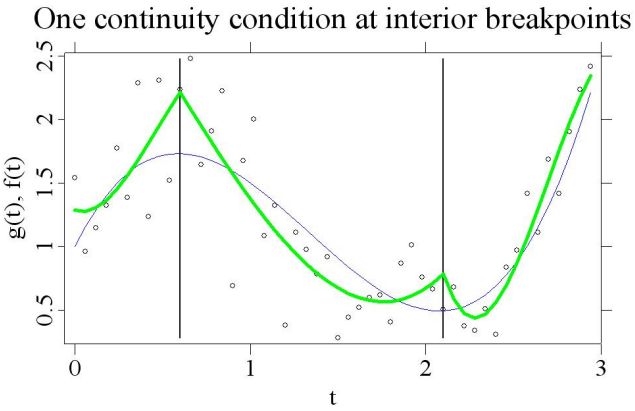


Figure 3.4: Continuous Piecewise Cubic Polynomials. (One continuity condition at each interior breakpoint.)

pcp1.xpl

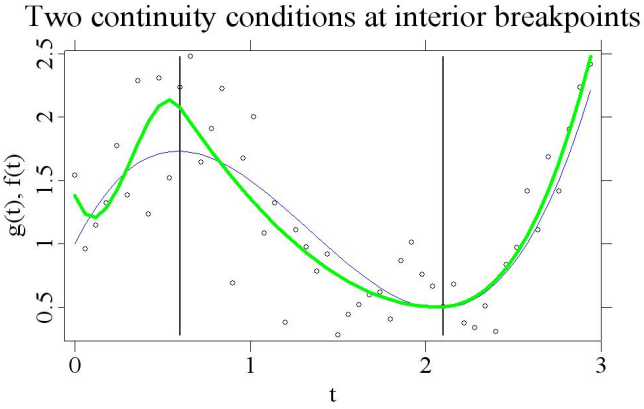


Figure 3.5: Continuous Piecewise Cubic Polynomials with continuous first derivative. (Two continuity conditions at each interior breakpoint.)

pcp1.xpl

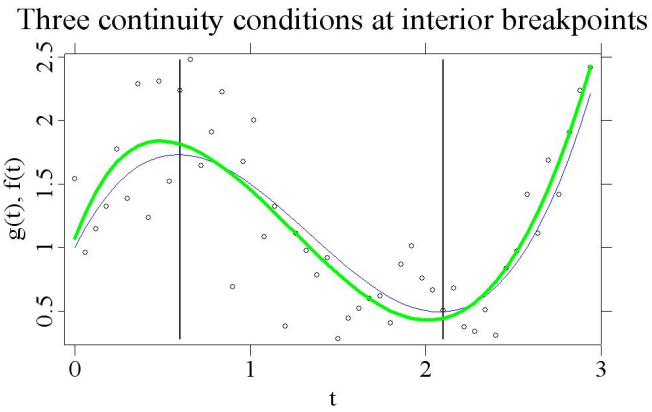


Figure 3.6: Continuous Piecewise Cubic Polynomials with continuous second derivative. (Three continuity conditions at each interior breakpoint.)

pcp1.xpl

truncated power basis.

Proposition 1 *The truncated power basis*

$$\phi_{ij}(t) = \begin{cases} (t-\xi_1)^j, & i = 1 \\ (t-\xi_i)_+^j, & i = 2,\ldots,m, \end{cases} \tag{3.18}$$

where

$$(t-\xi_i)_+^j = \begin{cases} (t-\xi_i)^j, & \text{if } t \geq \xi_i \\ 0, & \text{otherwise}, \end{cases} \tag{3.19}$$

and $j = \nu_i, \ldots, K-1$ *with* $\nu_1 \stackrel{\text{def}}{=} 0$ *is a basis of* $\mathcal{P}_{K,\xi,\nu}$.

Proof. de Boor (1978).

In order to maintain the flexibility while at the same time achieving some degree of global smoothness, one certain class of piecewise polynomial functions, the so called polynomial splines, has been introduced.

3.4.2 Polynomial Splines

When approximating by a piecewise polynomial function one often wants to achieve a globally smooth function. The concept of continuity conditions introduced in the

last subsection can be used to indicate a suitable subset of $\mathcal{P}_{K,\xi,\nu}$. This leads to the following definition:

Definition 7 *Let $K > 0$ be an integer indicating the order of piecewise polynomials, $\xi = \{\xi_i\}_{i=1}^{m+1}$ be a given breakpoint sequence with $t_L = \xi_1 < \xi_2 < \ldots < \xi_m < \xi_{m+1} = t_U$, and the vector $\nu = \{\nu_i\}_{i=2}^{m}$ counts the number of continuity conditions at each interior breakpoint.*

If $\nu_i = K - 1$ for all $i = 2, \ldots, m$ then the piecewise polynomial function $f(t)$ is called ***polynomial spline of order*** *K.*

Splines minimize average squared curvature and consequently perform better than interpolating polynomials. Furthermore, every continuous function on the interval $J = [t_L, t_U]$ can be approximated arbitrary well by polynomial splines with the order K fixed, given that a sufficient number of knots is provided. The approximation power of splines also holds for the simultaneous approximation of derivatives by derivatives of splines. The piecewise polynomial structures allows for easy and fast computation of splines and their derivatives.

Since polynomial splines are a special subset of piecewise polynomial functions they can also be written in terms of the truncated power basis introduced in Subsection 3.4.1.

For numerical application the truncated power basis (3.18) is not well suited. For example, if an argument t is near the upper bound of the interval J, it is necessary to evaluate all of the basis functions and to compute the entire sum. To reduce the computational complexity a basis with local support would be useful. By forming certain linear combinations of the truncated power functions a new basis for $P_{K,\xi,\nu}$ whose elements vanish outside a small interval can be obtained, as will be shown in the next subsection.

3.4.3 Definition and Properties of B-splines

Originally, B-splines has been defined as a divided difference of the truncated power basis. Divided differences arises out of the Newton form of interpolation, where for

$n+1$ pairs $\{t_i, y_i\}_{i=0}^{n}$ of arguments t_i and values of an underlying function f with $y_i = f(t_i)$ a polynomial is applied in the form of

$$\begin{aligned} p(t) &= c_0 + c_1(t-t_0) + c_2(t-t_0)(t-t_1) + \dots \\ &\quad + c_n(t-t_0)(t-t_1)\dots(t-t_n), \end{aligned} \tag{3.20}$$

with

$$\begin{array}{lcll} p(t_0) & = & c_0 & = y_0 \\ p(t_1) & = & c_0 + c_1(t_1 - t_0) & = y_1 \\ p(t_2) & = & c_0 + c_1(t_2 - t_0) + c_2(t_2 - t_0)(t_2 - t_1) & = y_2 \\ \text{etc.} & & & \end{array} \tag{3.21}$$

The coefficients $c_k, k = 0, \dots, n$ are uniquely determined by the $n+1$ pairs of arguments and function values. Let this relation be denoted by

$$c_k \stackrel{\mathbf{def}}{=} [x_0, x_1, \dots, x_k] f. \tag{3.22}$$

Schwarz (1997) shows that (3.22) can be computed recursively using

$$[x_{i_0}, \dots, x_{i_k}]\, f = \frac{[x_{i_1}, \dots, x_{i_k}]\, f - \left[x_{i_0}, \dots, x_{i_{k-1}}\right] f}{x_{i_k} - x_{i_0}}, k = 1, \dots, n, \tag{3.23}$$

where $i_0, \dots, i_k$ are pairwise different integers with $0 \le i_j \le n, j = 0, \dots, k$, and $[x_k]\, f = y_k, k = 0, \dots, n$. Because of the structure of (3.23), the term "divided difference" is used for (3.22). For a discussion of the main properties of divided differences see de Boor (1978).

Hence, we can now give the definition of (normalized) B-splines.

Definition 8 (Normalized B-splines) *For a nondecreasing sequence of knots τ the i-th normalized B-spline of order k is defined by the rule*

$$B_{i,k}(t) = (\tau_{i+k} - \tau_i)[\tau_i, \dots, \tau_{i+k}](\bullet - t)_+^{k-1}, \quad \forall \quad t \in \mathbb{R}. \tag{3.24}$$

Here the "placeholder" is used to indicate that the k-th divided difference of the function $(\tau - t)_+^{k-1}$ of the two variables τ and t is to be taken by fixing t and considering $(\tau - t)_+^{k-1}$ as a function of t alone, see de Boor (1978). In equation (3.24) the factor $(\tau_{i+k} - \tau_i)$ is a normalization factor designed to produce the identity

$$\sum_i B_{i,k}(t) = 1, \quad \forall \quad t \in J \tag{3.25}$$

which asserts that the B-splines form a partition of unity. In the following, for simplicity normalized B-splines are referred to as B-splines.

B-splines are in fact a basis of the space of piecewise polynomial functions. For the proof see de Boor (1978), Schumaker (1981), or Curry & Schoenberg (1966), respectively.

The dimension of a B-spline basis is greater than the dimension of a basis of splines. Therefore some additional knots have to be included. Schumaker (1981) called this extended partitioning. To extend the partitioning by a sequence of breakpoints $\{\xi_i\}_{i=1}^{m+1}$, $K-1$ additional initial and final knots are needed such that $\tau_1 \leq \tau_2 \leq \ldots \leq \tau_{K-1} \leq \xi_1$ and $\xi_{m+1} \leq \tau_{n+1} \leq \tau_{n+2} \leq \ldots \leq \tau_{n+K-1}$. The actual values of these additional knots are arbitrary. Also in FDA it is customary to make them all the same and equal to ξ_1 and ξ_{m+1}, respectively.

The interior knot sequence $\tau_K, \ldots, \tau_n$ must be defined out of the breakpoint sequence ξ. The desired amount of smoothness at an interior breakpoint $\xi_i, i = 2, \ldots, m$ as measured by the number of continuity conditions ν_i must be translated into a consistent multiplicity of knot $\tau_\bullet$ following the relationship

$$K - \nu_i = \text{number of knots at } \xi_i.$$

In other words, for each additional knot at a point ξ_i, the spline function will have one less continuous derivative at that knot. A B-spline of order K with K knots at a breakpoint can, but not must, be discontinuous at that point.

Figures 3.7-3.10 show sequences of B-splines up to order four with equidistant knots from 0 to 10. The partitions are extended appropriately.

Figure 3.11 illustrates the ability of B-splines to respond to different amounts of smoothness at the interior breakpoints.

Using (3.24) for numerical computation can be problematic because precision might have been lost during the computation of the various quotients needed for divided differences. Also, special provisions have to be made in the case of repeated or multiple knots. Such a calculation would amount to evaluate $B_{i,k}(x)$ in

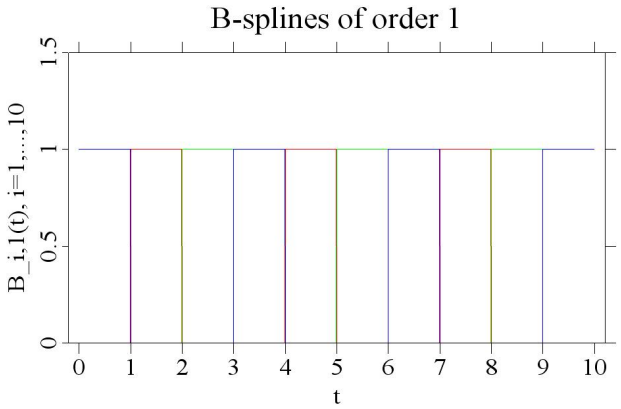


Figure 3.7: 1st order B-splines with (extended) partition $\tau = \{0, 1, 2, \ldots, 9, 10\}$.

`bspl1.xpl`

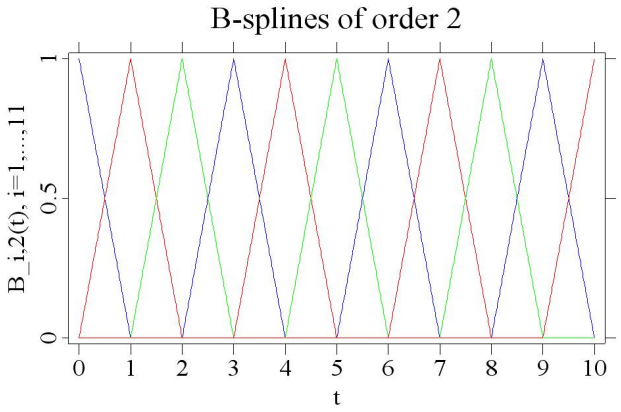


Figure 3.8: 2nd order B-splines with extended partition $\tau = \{0, 0, 1, 2, \ldots, 9, 10, 10\}$.

`bspl2.xpl`

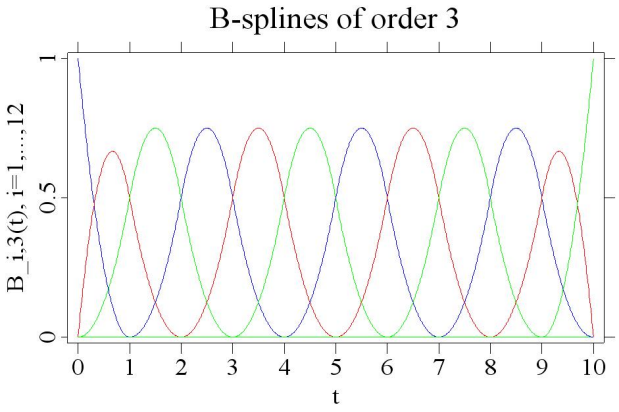


Figure 3.9: 3rd order B-splines with extended partition $\tau = \{0, 0, 0, 1, 2, \ldots, 9, 10, 10, 10\}$.

bspl3.xpl

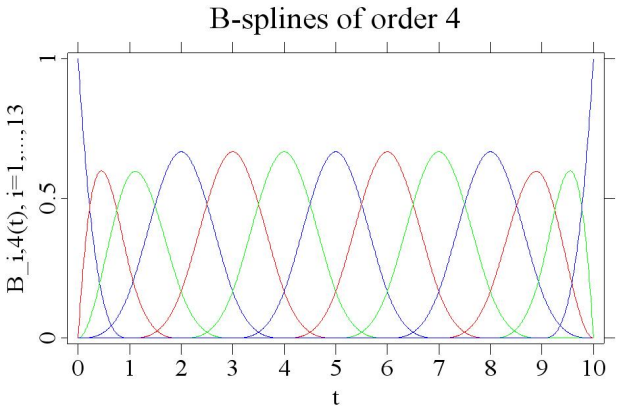


Figure 3.10: 4th order B-splines with extended partition $\tau = \{0, 0, 0, 0, 1, 2, \ldots, 9, 10, 10, 10, 10\}$.

bspl4.xpl

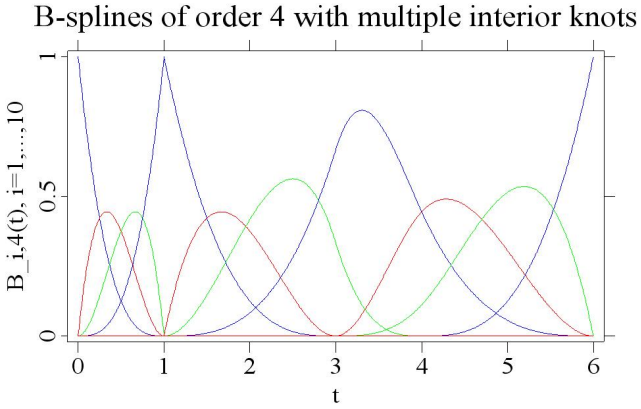


Figure 3.11: B-splines of order 4 with knot sequence $\tau = \{0, 0, 0, 0, 1, 1, 1, 3, 3, 4, 6, 6, 6, 6, 6\}$.

bspl5.xpl

terms of the truncated power basis and would therefore be beset with precisely the difficulties which was hoped to be avoided by introducing B-splines.

Fortunately, due to de Boor (1978) it is possible to evaluate B-splines with the help of a recurrence relation which requires no special arrangements in the case of multiple knots and does not suffer unnecessary loss of precision.

Proposition 2 *The i-th (normalized) B-spline of order k* $(k = 1, \ldots, K)$ *for the nondecreasing knot sequence* $\tau = \{\tau_i\}_{i=1}^{n+K}$ *can be computed as*

$$B_{i,k}(t) = \frac{t - \tau_i}{\tau_{i+k-1} - \tau_i} B_{i,k-1}(t) + \frac{\tau_{i+k} - t}{\tau_{i+k} - \tau_{i+1}} B_{i+1,k-1}(t), \tag{3.26}$$

where

$$B_{i,1}(t) = \begin{cases} 1 & \text{if } \tau_i \leq t < \tau_{i+1} \\ 0 & \text{otherwise,} \end{cases} \tag{3.27}$$

and $i = 1, \ldots, n + K - k, \quad n = mK - \sum_{i=2}^{m} \nu_i$.

Proof. de Boor (1978).

In (3.26) each B-spline $B_{i,k}(x)$ is computed from two B-spline basis functions of degree $k - 1$. Hence, $B_{i,k}(x)$ is recursively built from basis functions of degree

$k = 1$. This is numerically stable since only convex combinations of nonnegative quantities are involved. Furthermore, $B_{i,k}(x) \geq 0$, for all i.

Since B-splines could also have multiple knots some care has to be taken when using the recurrence relation (3.26) in order to avoid division by zero. By (3.27) $B_{i,1}(t) = 0$ if $\tau_i = \tau_{i+1}$. By induction one can show that $B_{i,k-1}(t) = 0$ if $\tau_i = \tau_{i+1} = \ldots = \tau_{i+k-1}$. Hence, the according term in (3.26) will be zero no matter how its coefficient will be defined.

B-splines have compact and local support, e.g. $B_{i,k}(x) = 0$, for $x \notin [\tau_i, \tau_{i+k}]$. This leads to a band structured cross product matrix and an $O(k)$ computation of all smoothing values, where $O(\bullet)$ denotes the order of computational complexity. Because of their local support, there are at most k non-zero basis functions of degree k on $[\tau_i, \tau_{i+k}]$.

When using Kth order B-splines with $m+1$ knots, where m denotes the number of truncated subintervals of J (see Subection 3.4.1) the basis expansion of the ith replication is

$$X_i(t) = \sum_{j=1}^{m+K-1} c_{ij}\phi_j(t), \tag{3.28}$$

where $\phi_j(t) = B_{j,K}(t)$. According to de Boor (1978), expansion (3.28) is also called a spline or B-spline function. The main advantage for computation comes from the local support property of B-splines. For $\tau_l \leq t \leq \tau_{l+1}, l \in [K, n]$ it is sufficient to calculate

$$X_i(t) = \sum_{j=l-K+1}^{l} c_{ij}\phi_j(t) = \sum_{j=l-K+1}^{l} c_{ij}B_{j,K}(t) \tag{3.29}$$

because all other B-splines of order K have no support on $[\tau_l, \tau_{l+1}]$.

3.4.4 Derivatives of B-splines

Equivalently to actual B-splines a recurrence relation could also be found for computing their mth derivative, as is claimed in the following proposition.

Proposition 3 *The mth derivative* $(m < K)$ *of the kth order B-spline can be computed recursively by*

$$D^m B_{i,k}(t) = (k-1)\left\{\frac{D^{m-1}B_{i,k-1}(t)}{\tau_{i+k-1}-\tau_i} - \frac{D^{m-1}B_{i+1,k-1}(t)}{\tau_{i+k}-\tau_{i+1}}\right\}. \tag{3.30}$$

Proof. Using (3.24) we get

$$DB_{i,k}(t) = -(k-1)(\tau_{i+k}-\tau_i)[\tau_i,\ldots,\tau_{i+k}](\bullet - t)_+^{k-2}. \tag{3.31}$$

It holds that

$$[\tau_i,\ldots,\tau_{i+k}](\bullet - t)_+^{k-2} = \frac{[\tau_{i+1},\ldots,\tau_{i+k}](\bullet - t)_+^{k-2} - [\tau_i,\ldots,\tau_{i+k-1}](\bullet - t)_+^{k-2}}{\tau_{i+k}-\tau_i}, \tag{3.32}$$

and using (3.24) in (3.32), (3.31) becomes

$$DB_{i,k}(t) = (k-1)\left\{\frac{B_{i,k-1}(t)}{\tau_{i+k-1}-\tau_i} - \frac{B_{i+1,k-1}(t)}{\tau_{i+k}-\tau_{i+1}}\right\}.$$

The first derivative of a B-spline is another B-spline of one less order. Recursively applying this technique one can compute higher order derivatives, since

$$D^m B_{i,k}(t) = D\left\{D^{m-1}B_{i,k-1}(t)\right\}.$$

□

Figures 3.12 and 3.13 illustrate $DB_{i,3}(t)$, and $D^2B_{i,3}(t)$ of a third order B-spline with knot sequence $\tau = \{0,2,3,6,10\}$, respectively. A B-spline of order k is a piecewise polynomial function of order k. Hence, it is obvious that its mth derivative $(m < k)$ must be a piecewise polynomial function of order $k-m$.

Since both recurrence relations (3.30) and (3.27) have the same structure and furthermore B-splines of order $K-m$ are needed for derivatives, their computation in one algorithm might be reasonable. This will also be the spirit of the C++ program `bspline.cpp` which is used for the quantlet `Bsplineevalgd.xpl` as described in Section 3.4.6.

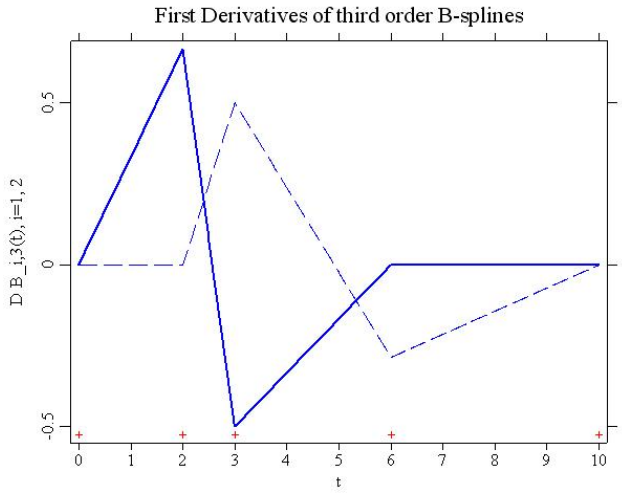


Figure 3.12: $DB_{1,3}(t)$ (solid line) and $DB_{2,3}(t)$ (dotted line) with $\tau = \{0, 2, 3, 6, 10\}$. The derivatives are piecewise polynomial functions of order 2.

bsplderiv1.xpl

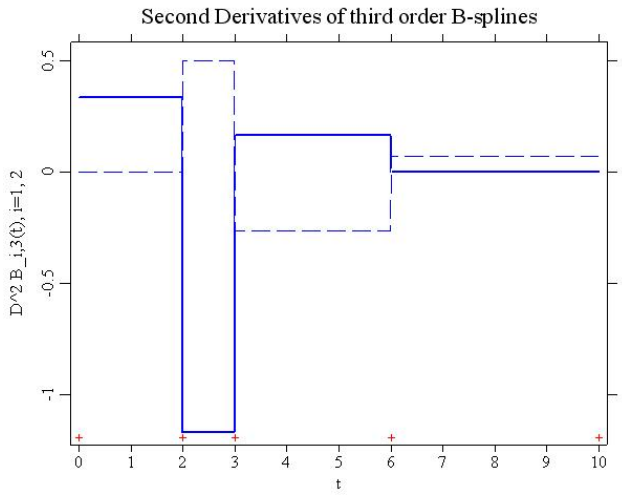


Figure 3.13: $D^2B_{1,3}(t)$ (solid line) and $D^2B_{2,3}(t)$ (dotted line) with $\tau = \{0, 2, 3, 6, 10\}$. The second order derivatives are piecewise constant functions, that means piecewise polynomial functions of order 1.

bsplderiv2.xpl

3.4.5 Tensor B-splines

In the case of multivariate replications there is a need for a generalization of the B-spline concept to the multidimensional case. Using the tensor-product approach is one reasonable possibility, because many of the simple algebraic properties of ordinary one-dimensional B-splines can be carried over, and fortunately because of its nature it is possible to work with the usual one-dimensional B-splines. Only a short overview on the main results can be given here. For a more detailed discussion the reader is referred to Schumaker (1981).

Definition 9 *Consider a given d-dimensional nondecreasing knot sequence* $\tau = \{\tau_{ij}\}_{i=1}^{n_j+K_j}$, $j = 1, \ldots, d$, $n_j = m_j K_j - \sum_{l=2}^{m} \nu_{jl}$ *where* m_j *and* K_j *denote the number of subintervals and order of the* j*-th dimension, respectively. The tensor-product B-splines is defined by*

$$B_{i_1 \ldots i_d, k_1 \ldots k_d}(t_1, \ldots, t_d) = \prod_{j=1}^{d} B_{i_j, k_j}(t_j). \tag{3.33}$$

General properties of the tensor-product B-splines can be derived from the corresponding properties of the one-dimensional B-splines. See Schumaker (1981) for discussion. Figure 3.14 and Figure 3.15 illustrate bivariate tensor-product B-splines.

Equivalently to (3.28) the basis expansion of the ith replication has the form

$$X_i(t_1, \ldots, t_d) = \sum_{r_1=1}^{m_1+K_1-1} \cdots \sum_{r_d=1}^{m_d+K_d-1} c_{i, r_1 \ldots r_d} B_{r_1 \ldots r_d, k_1 \ldots k_d}(t_1, \ldots, t_d) \tag{3.34}$$

and because of the inherited property of local support for $\mathbf{t} = (t_1, \ldots, t_d) \in \bigotimes_{l=1}^{d} [\tau_{j_l, l}, \tau_{j_l+1, l}]$ it is sufficient to compute

$$X_i(\mathbf{t}) = \sum_{r_1=j_1-K_1}^{j_1} \cdots \sum_{r_d=j_d-K_d}^{j_d} c_{i, r_1 \ldots r_d} B_{r_1 \ldots r_d, k_1 \ldots k_d}(t_1, \ldots, t_d) \tag{3.35}$$

for $j_r \in [K_r, n_r], r = 1, \ldots, d$. For functional data tensor B-splines will be primarily necessary when computing bifd objects, such as covariances.

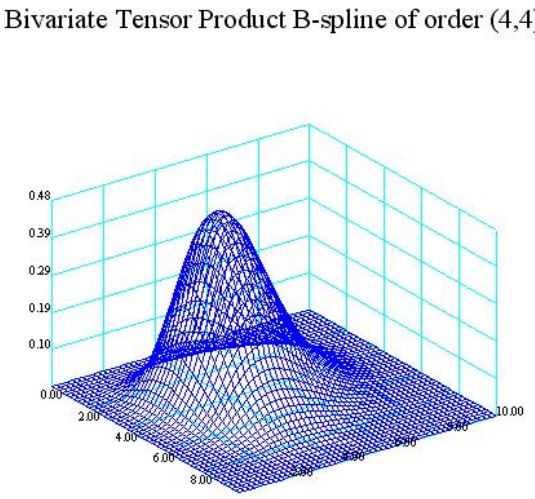


Figure 3.14: A bivariate tensor-product B-spline with $k_1 = k_2 = 4$ and $\tau_{\bullet 1} = \tau_{\bullet 2} = \{0, 2, 3, 6, 10\}$.

`tensor1.xpl`

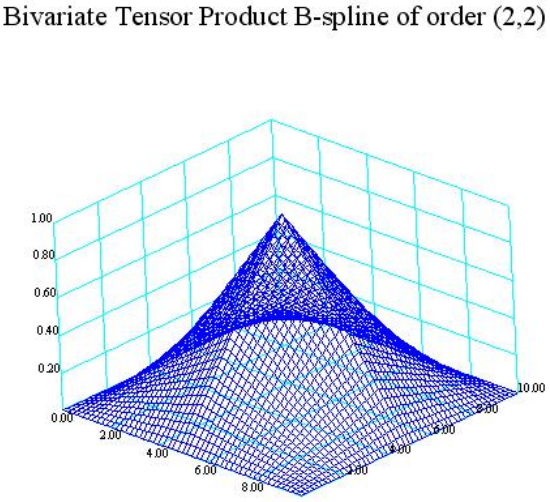


Figure 3.15: A bivariate tensor-product B-spline with $k_1 = k_2 = 2$ and $\tau_{\bullet 1} = \tau_{\bullet 2} = \{0, 5, 10\}$.

`tensor2.xpl`

3.4.6 XploRe Quantlet for B-spline Bases

```
phi = Bsplineevalgd (tvec, norder{, xvec{, deriv{, boolextend}}})
      computes the basis matrix of B-splines for a given non-decreasing
      sequence of knots tvec
```

The quantlet `Bsplineevalgd` allows to compute B-spline bases and one or more of their derivatives simultaneously for a matrix of arguments. `Bsplineevalgd` uses the C++ based dynamically linked library `bspline.dll`. The syntax of the quantlet is

```
phi = Bsplineevalgd (tvec, norder{, xvec{, deriv{, boolextend}}})
```

with input parameters

`tvec` - a $(n \times 1)$ vector of strictly non-decreasing knot sequence. The same knot sequence will be applied to all arguments `xvec`.

`norder` - integer indicates the B-spline order (e.g. for cubic B-splines `norder` $= 4$)

`xvec` - a $(p \times q)$ matrix of arguments where the B-splines matrix should be evaluated at. The default value is `tvec`.

`deriv` - a $(r \times 1)$ vector of integers (≥ 0) containing the orders of derivatives to compute. If `deriv` $= 0$ the actual B-spline bases will be calculated. The default value is `deriv` $= 0$.

`boolextend` - If `boolextend` $= 1$ tvec will be extended appropriately to get multiple exterior knots. The default value is `boolextend` $= 1$.

This quantlet returns a $(p \times$ `norder` $\times r \times q)$ array, where one row contains the evaluated bases for one argument.

The automatic extension of the given knot sequence `tvec` can be suppressed by using `boolextend` $= 0$ as input parameter. In this case the given knot sequence will be used directly. Furthermore, this quantlet is able to deal with multiple interior

knots, that means the only restriction is that the knot sequence must be strictly non-decreasing.

When computing tensor product B-splines the same knot sequence must be applied to all dimensions. Furthermore, the orders of derivative cannot vary between the dimensions. Nevertheless, this is sufficient to compute the most common bifd objects, such as covariance.

In order to get non-singular basis matrices all function and derivative values of arguments equal to the upper exterior knots must be set equal to its left-sided limes. Numerically, this will be done by subtracting the smallest possible amount of 10^{-15} from these arguments. As a result, their basis function values have an absolute numerical error of at most 10^{-15}.

3.5 Using fdbasis Objects in XploRe

`fdbasis = createfdbasis(type,rangeval,nbasis,params)`
creates the fdbasis object

`basismat = getbasismatrix (evalarg, fdbasis {, Lfd})`
computes the basis matrix, its derivatives, or applied LDOs in `evalarg` associated with `fdbasis` object

The quantlet `createfdbasis` uniquely defines a basis by collecting all information necessary to compute the basis. The quantlet has the syntax

```
fdbasis = createfdbasis (type, rangeval, nbasis, params)
```

with input parameters

`type` - a string indicating the type of basis. This may be one of "fourier", "bspline", "poly"

`rangeval` - a vector of length 2 containing the lower and upper boundaries for the range of argument values

`nbasis` - the number of basis functions

`params` - Dimension and contents depend on the basis type. If the basis is "fourier", this is a single number indicating the period. If the basis is "bspline", params is strictly nondecreasing sequence of knots. If the basis is "poly", params can optionally obtain a constant shift parameter.

This quantlet creates a functional data basis (fdbasis) object in two steps. First it recognizes the type of basis by use of several variant spelling, and afterwards it sets up the functional data basis depending on the type.

Technically, the fdbasis object is a list, consisting of the four input parameters. Introducing a quantlet such as `createfdbasis` helps to avoid complications. Furthermore, it offers the possibility to identify misspecifications at a very early stage. The already implemented basis types just need list elements which are of elementary types, such as strings and vectors. It might be in implementing another basis that the params element requires a list. In such a case, an appropriate list may be created before `createfdbasis` is called.

For evaluating the basis functions, its derivatives, or applied linear differential operators at a given vector, respectively matrix, of arguments the quantlet `getbasismatrix` can be used. Its syntax is

```
basismat = getbasismatrix (evalarg, fdbasis {, Lfd})
```

with input parameters

`evalarg` - a $(p \times q)$ matrix of of values at which all functions are to be evaluated.

`fdbasis` - an fdbasis object

`Lfd` - This can be a $(r \times 1)$ vector of integers which indicate the order of derivatives to obtain. In the case of an LDO with constant coefficients Lfd must be a $(r \times 2)$ matrix, where the first column contains the coefficients, the second one the orders of derivatives. When to apply an LDO with variable coefficients Lfd must be an LDO object.

This quantlet returns an $(p \times$ `fdbasis.nbasis` $\times x \times q)$ array containing the functional values, where $x = r$ if `Lfd` is a vector, and $x = 1$ if `Lfd` is a $(r \times 2)$ matrix, or an LDO object. `fdbasis.nbasis` is the element of the fdbasis object which contains the number of basis functions. Directly this quantlet will be used rarely. Nevertheless, it is frequently referred to when dealing with fd objects.

The quantlet `getbasismatrix` supports four different `Lfd` input types:

1. one single derivative,
2. a vector of derivatives,
3. an LDO with constant coefficients,
4. an LDO with variable coefficients.

In case (1) `Lfd` is a single integer, in case (2) `Lfd` is a vector of integers, in case (3) `Lfd` is $(r \times 2)$ matrix, where the first column contains the coefficients and the second the orders of derivative, in case (4) `Lfd` is an LDO object, that means a list containing two elements: `ldoname` which indicates the name of the `proc-endp` environment where the `LDO` is defined and `derivs` the vector of orders of derivatives needed for the LDO. This LDO object can also be created by the quantlet `createLDO`.

Some technical problems arise out of the manifold nature of the `Lfd` input parameter. `Lfd` can be a matrix, or a list, respectively, what the quantlet has to check for. The implemented algorithm for that purpose might be worth to be mentioned. First a list is created out of the `Lfd` input parameter, no matter whether or not it is already a list. Then for the created list the number of rows of the vector containing the dimensions is computed. If this is unequal to one, then `Lfd` is a matrix, and hence, an LDO with constant coefficients. If this is one and unequal to the size of the created list, then `Lfd` must be a LDO object. Otherwise `Lfd` is a vector, and contains the order of derivatives.

If some other bases should be added, then a link to their evalgd quantlet must be added in the `switch-case` environment of `getbasismatrix`.

The fdbasis objects only contain information on the type of basis expansion. Its coefficients has been left open yet to be estimated. How this can be done is described in detail in the next chapter.

Chapter 4

Smoothing Methods for Functional Data

In Chapter 3 a functional observation $X_i(t)$ was represented by a linear combination of K basis functions as

$$X_i(t) = \Phi(t)\mathbf{c_i}, \quad , t \in J, \quad \mathbf{c_i} \in \mathbb{R}^K, \quad i = 1, \ldots, n. \tag{4.1}$$

The vector of coefficients $\mathbf{c_i}$ must be estimated with the help of the sample data $(\mathbf{t_i}, \mathbf{y_i})$ and the relation

$$\mathbf{y_i} = X_i(\mathbf{t_i}) + \epsilon_\mathbf{i} = \Phi(\mathbf{t_i})\mathbf{c_i} + \epsilon_\mathbf{i}. \tag{4.2}$$

This coincides with the classical linear regression model, and the most obvious idea for estimating the coefficients $\mathbf{c_i}$ is by minimizing the least squares criterion to obtain the generalized least squares (GLS) estimator. This yields

$$\hat{\mathbf{c}}_\mathbf{i} = \left\{ \Phi(\mathbf{t_i})^\top \Sigma_i^{-1} \Phi(\mathbf{t_i}) \right\}^{-1} \Phi(\mathbf{t_i})^\top \Sigma_i^{-1} \mathbf{y_i} \tag{4.3}$$

as an estimator for $\mathbf{c_i}$. If the covariance matrix Σ_i of the vector of error terms $\epsilon_\mathbf{i}$ is unknown, an estimator $\hat{\Sigma}_i$ must be used. This turns the procedure into the estimated generalized least squares (EGLS) method. For a discussion as well as estimating procedures for an unknown covariance matrix see Judge et al. (1988).

Unfortunately, the (E)GLS estimator leaves all the control on smoothness of the underlying function to the basis functions. Therefore controlling for smoothness

can only be done by choosing an appropriate basis expansion and in a discrete way by specifing a sufficient number of basis functions. One possibility to overcome this lack is to use the roughness penalty approach which will be introduced in the next section.

4.1 Penalized Regression

4.1.1 The Roughness Penalty Approach

The roughness penalty approach is an extension of generalized least squares. The main idea comes from the fact that the mean squared error (MSE) can be decomposed into

$$MSE = Bias^2 + Variance,$$

where $Bias^2$ is defined as the sum of squared errors. Note that in minimizing MSE, it can often greatly reduced by trading a little bit bias off against a lot of sampling variance. According to the underlying model $\mathbf{y_i} = X_i(\mathbf{t_i}) + \epsilon_\mathbf{i}$, a completely unbiased estimate of the function can be produced by a curve fitting $\mathbf{y_i}$ exactly, as by Newton interpolation, see Subsection 3.4.3. But any such curve must have high variance, manifested in the rapid local variation of the curve. Hence, a measure of its variance is the concept of roughness, as introduced in Section 2.2.

The idea of the roughness penalty approach is to penalize functions that would lead to an estimated rough function. This will be done by determining the coefficients in the usual GLS way but to introduce an additional penalty term. The penalized least squares criterion for the ith replication is

$$PENSSE_\lambda(X_i|\mathbf{y_i}) = \{\mathbf{y_i} - X_i(\mathbf{t_i})\}^\top \mathbf{W_i} \{\mathbf{y_i} - X_i(\mathbf{t_i})\} + \lambda PEN_L(X_i), \qquad (4.4)$$

where λ is a smoothing parameter that measures the rate of exchange between the fit to the data and variability of the estimated function, $\mathbf{W_i}$ is a symmetric weight matrix, and the roughness PEN_L is used as the penalty term. Minimizing (4.4) with respect to $X_i(t)$ leads to an estimated function $\hat{X}_i(t), t \in J$. The classical roughness penalty approach uses $L = D^2$ as penalty operator, but for the same

reasons as discussed in Section 2.2 when roughness of functions was introduced, it might be useful to generalize this concept. Reinsch (1967) shows that when $L = D^2$ the resulting function $\hat{X}_i(t)$ is a cubic spline with knots at the data points $\mathbf{t_i}$. When using a general LDO Kimeldorf & Wahba (1970) show that the resulting functions are so called L-splines. See Schumaker (1981) for a formal definition and the properties of L-splines.

Figure 4.1 illustrates the impact of different amounts of λ on the estimated function, using a B-spline basis expansion and $L = D^2$ as penalty operator. As $\lambda \to 0$, roughness matters less and the estimated function $\hat{X}_i(t)$ tends to interpolate the underlying data. As $\lambda \to \infty$, roughness will be strongly penalized and as a result $\hat{X}_i(t)$ tends to be hypersmooth, here that means it tends to be a linear function because of the given penalty operator.

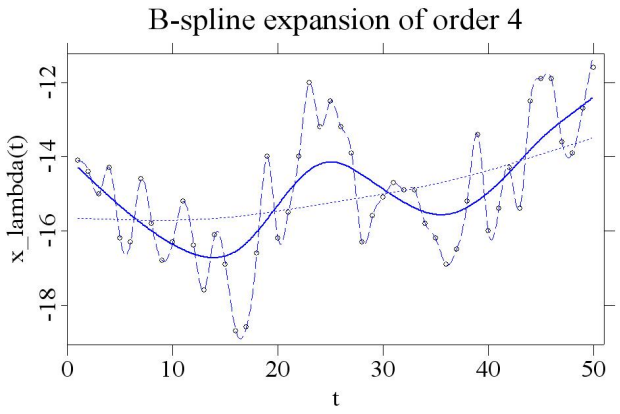


Figure 4.1: Example of B-spline expansions of order 4 with $\lambda = 0.00001$ (dashed), $\lambda = 100$ (solid), and $\lambda = 10000$ (dotted) for Canadian weather data.

`rp1.xpl`

When $X_i(t)$ is represented by the basis expansion

$$X_i(t) = \sum_{k=1}^{K} c_{ik}\phi_k(t) = \Phi(t)\mathbf{c_i}, \tag{4.5}$$

the penalty term can be written

$$\begin{aligned} PEN_L(X_i) &= \int_J L\left\{\Phi(t)\mathbf{c_i}\right\}^\top L\left\{\Phi(t)\mathbf{c_i}\right\} dt \\ &= \mathbf{c_i}^\top \int_J L\Phi(t)^\top L\Phi(t) dt \mathbf{c_i} \\ &= \mathbf{c_i}^\top \mathbf{R}\mathbf{c_i}, \end{aligned} \tag{4.6}$$

where the order K symmetric matrix $\mathbf{R}$ contains elements

$$\mathbf{R}_{kl} = \int_J L\phi_k(t) L\phi_l(t) dt = \langle L\phi_k, L\phi_l \rangle, k, l = 1, \ldots, K.$$

How to compute the matrix of inner products $\mathbf{R}$ will be shown in Section 4.2 .

Using the basis function expansion (4.5) and (4.6) the penalized least squares criterion becomes

$$PENSSE_\lambda(X_i|\mathbf{y_i}) = \left\{\mathbf{y_i} - \Phi(\mathbf{t_i})\mathbf{c_i}\right\}^\top \mathbf{W_i} \left\{\mathbf{y_i} - \Phi(\mathbf{t_i})\mathbf{c_i}\right\} + \lambda \mathbf{c_i}^\top \mathbf{R}\mathbf{c_i}. \tag{4.7}$$

If the error terms $\epsilon_\mathbf{i}$ are assumed to be heteroscedastic, then $\mathbf{W_i}$ becomes the inverse of the covariance matrix Σ_i, respectively the estimated covariance matrix $\hat{\Sigma}_i$ if the variances of the error term are unknown. $\mathbf{W_i}$ can also depend on the argument t as will be discussed in Section 4.4.

The vector $\mathbf{c_i}$ is minimized by

$$\hat{\mathbf{c}}_\mathbf{i} = \left\{\Phi(\mathbf{t_i})^\top \mathbf{W_i} \Phi(\mathbf{t_i}) + \lambda R\right\}^{-1} \Phi(\mathbf{t_i})^\top \mathbf{W_i} \mathbf{y_i}. \tag{4.8}$$

The data fitting vector $\hat{\mathbf{y}}_\mathbf{i} = X_i(\mathbf{t_i})$ is

$$\hat{\mathbf{y}}_\mathbf{i} = \Phi(\mathbf{t_i}) \left\{\Phi(\mathbf{t_i})^\top \mathbf{W_i} \Phi(\mathbf{t_i}) + \lambda \mathbf{R}\right\}^{-1} \Phi(\mathbf{t_i})^\top \mathbf{W_i} \mathbf{y_i} = S_{\phi,\lambda,\mathbf{i}} \mathbf{y_i}, \tag{4.9}$$

where the smoothing matrix

$$S_{\phi,\lambda,\mathbf{i}} = \Phi(\mathbf{t_i}) \left\{\Phi(\mathbf{t_i})^\top \mathbf{W_i} \Phi(\mathbf{t_i}) + \lambda \mathbf{R}\right\}^{-1} \Phi(\mathbf{t_i})^\top \mathbf{W_i} \tag{4.10}$$

maps the data into the fit.

As shown above, the roughness penalty estimator has a structure similar to the GLS estimator. Moreover, it provides a natural and flexible approach to curve

estimation, especially when representing functional data as finite basis expansions. Two parameters must be estimated in a way before starting the roughness penalty approach: the smoothing parameter λ, and if necessary the linear differential operator used for penalization. How to do that is described in the next two subsections.

4.1.2 Choosing the Smoothing Parameter

Green & Silverman (1994) distinguish two philosophical approaches to the question of choosing the smoothing parameter. The first approach is to choose the smoothing parameter λ subjectively. By varying λ features of the data that arise on different "scales" can be explored, and the one parameter value which "looks best" might be chosen. The second approach is to use an automatic method of choice such as cross validation. Especially in the functional data context where large data sets on several replications are used, an automatic method is essential in order to get comparable results. Furthermore an automatic choice can in any case be used as a starting point for subsequent subjective adjustment, see Silverman (1985).

Probably the most attractive class of methods for choosing the smoothing parameter is cross-validation. The main idea is to set aside a subset of the data, the validation sample, to fit the model to the rest of the data, then to assess the fit to the validation sample, and finally to choose the λ value that gives the best fit. An important early reference to the use of cross-validation to guide the choice of smoothing parameter is Craven & Wahba (1979). In the functional case, we omit the functional data one at a time, and so the various terms in the cross-validation score relate to the way that a whole function $X_i(t)$ is predicted from the other functions in the data set. The idea of leaving out whole data curves is discussed by Rice & Silverman (1991).

Following Ramsay & Silverman (2002), let $m_\lambda^{-i}(t)$ denote the smoothed sample mean calculated with smoothing parameter λ from all replications except $X_i(t)$. As a measure of global discrepancy compute the integrated squared error (ISE)

$$ISE\{m_\lambda^{-i}(t)\} = \int_J \{m_\lambda^{-i}(t) - X_i(t)\}^2 dt, \tag{4.11}$$

to see how well $m_\lambda^{-i}(t)$ predicts $X_i(t)$. The cross-validation score $CV(\lambda)$ is computed by summing up the integrated squared errors (4.11) over all n replications

$$CV(\lambda) = \sum_{i=1}^{n} \int_J \{m_\lambda^{-i}(t) - X_i(t)\}^2 dt. \tag{4.12}$$

The smaller the value of $CV(\lambda)$, the better the performance of λ as measured by the cross-validation score. Hence, minimizing (4.12) with respect to λ gives the optimal smoothing parameter.

Since cross-validation is very time consuming, and tends too often to undersmooth the data, Eubank (1988) introduces the generalized cross-validation criterion. Here, the value of λ is chosen to minimize

$$GCV(\lambda) = \frac{n\text{SSE}(\lambda)}{(n - df_\lambda)^2} = \frac{n}{n - df_\lambda}\hat{\sigma}^2(\lambda), \tag{4.13}$$

where $SSE(\lambda) = \{\mathbf{y_i} - \Phi_i(\mathbf{t_i})\hat{\mathbf{c}}_\mathbf{i}\}^\top \mathbf{W_i}\{\mathbf{y_i} - \Phi_i(\mathbf{t_i})\hat{\mathbf{c}}_\mathbf{i}\}$ and $\hat{\sigma}^2(\lambda) = SSE(\lambda)/(n - df_\lambda)$. df_λ is the effective number of parameters or degrees of freedom that $\hat{X}_i(t) = \Phi_i(t)\hat{\mathbf{c}}_\mathbf{i}$ uses in estimating $X_i(t)$. Heckman & Ramsay (2000) define

$$df_\lambda = \text{tr}(S_{\phi,\lambda,\mathbf{i}}) + \text{number of estimated parameters}, \tag{4.14}$$

where $S_{\phi,\lambda,\mathbf{i}}$ is the smoothing matrix as before. One can show that, for fixed coefficients in the LDO, $\text{tr}(S_{\phi,\lambda})$ is a decreasing function of λ ranging from n_i when $\lambda = 0$ to the dimension of the kernel of L, as $\lambda \to \infty$, see Heckman & Ramsay (2000).

Usually, the numerator of GCV is small (i.e., $\Phi_i(\mathbf{t_i})\hat{\mathbf{c}}_\mathbf{i}$ is close to interpolating the data) when the denominator is small (when df_λ is close to n_i). Thus minimizing GCV means fitting the data well with few parameters, see Heckman & Ramsay (2000).

4.1.3 Choosing the Linear Differential Operator

As introduced in Definition 3, for using a linear differential operator (LDO) its order m and all (variable) coefficients $w_j(t), j = 0, \ldots, m-1$ must be known. In practice, there is usually a general idea of the order and the coefficients because

Operator L	Parametric family for the kernel of L
D^2	$\{1, t\}$
D^4	$\{1, t, t^2, t^3\}$
$D^2 + \gamma D, \gamma \neq 0$	$\{1, \exp(-\gamma t)\}$
$D^4 + \omega^2 D^2, \omega \neq 0$	$\{1, t, \cos(\omega t), \sin(\omega t)\}$
$(D^2 - \gamma D)(D^2 + \omega^2 D), \gamma, \omega \neq 0$	$\{1, \exp(\gamma t), \cos(\omega t), \sin(\omega t)\}$
$D - \omega(\cdot)I, w(t) \neq 0$	$\left\{\exp\left[\int w(u)du\right]\right\}$
$D^2 - w(\cdot)D, w(t) \neq 0$	$\left\{1, \int \exp\left[\int w(v)dv\right] du\right\}$

Table 4.1: Examples of differential operators and bases for the corresponding parametric families. Source: Heckman & Ramsay (2000).

many models are based upon scientific or physical models, and any of these can be considered as the favored model for LDO. Table 4.1 summarizes examples of linear differential operators and bases for the corresponding parametric families.

If the coefficients are not known they must be estimated out of the data. Usually, one uses a parametric model and assumes $w_j(t|\theta)$ to depend on some parameter vector θ. For example, when dealing with periodic data θ may include an unknown period. The parameter vector can be estimated by using nonlinear least squares techniques to minimize

$$\min_{\alpha_j, \theta} \sum_{i=1}^{n} \{\mathbf{y_i} - \sum_j \alpha_j w_j(\mathbf{t_i}|\theta)\}^2, \tag{4.15}$$

where α_j are some additional weight factors. Heckman & Ramsay (2000) also discusses the possibility of minimizing the GCV criterion with respect to both λ and the parameter vector θ.

However, the parametric model must in a way fit the data. Otherwise there would not be any gain from using the associated penalty.

4.2 Computing Inner Products

For the roughness penalty approach the penalty term

$$PEN_L(X_i) = \|LX_i\|^2 = \langle LX_i, LX_i \rangle \tag{4.16}$$

must be computed. Using the basis function expansion (4.1) for $X_i(t)$ the penalty term can be written as

$$PEN_L(X_i) = \mathbf{c_i}^\top \mathbf{R} \mathbf{c_i} \tag{4.17}$$

with

$$\mathbf{R}_{kl} = \int_J L\phi_k(t) L\phi_l(t) dt = \langle L\phi_k, L\phi_l \rangle, k, l = 1, \ldots, K$$

as shown in (4.6). Hence, the most critical part in computing $PEN_L(X_i)$ is obviously the calculation of the inner product matrix $\mathbf{R}$. How to do this in particular depends on the underlying basis function and the applied linear differential operator L. The main analytic and numerical algorithms are presented in Subsection 4.2.1. The accompanied quantlet `inprod` is introduced in Subsection 4.2.2.

4.2.1 Analytic and Numerical Solutions

If the linear differential operator L has constant coefficients then it is possible in some cases to compute R analytically. Since in the roughness penalty context inner products are only needed for two basis functions of the same family and with the same order of derivatives we will restrict the presented analytic solutions to these cases. In all other cases, the use of a numerical algorithm might be sufficient. We will present the Romberg integration approach for that reason later on in this subsection.

For computing inner products of Fourier bases the following proposition is useful.

Proposition 4 *If $\phi_l(t), \phi_k(t), l, k = 0, \ldots, K$ are Fourier bases and the period is $T = t_U - t_L$ then for $m = 0, 1, 2, \ldots$*

1. $\int_{t_L}^{t_U} D^m\phi_0(t) D^m\phi_0(t) dt = \begin{cases} 1, & m = 0, \\ 0, & m > 0, \end{cases}$

2. $\int_{t_L}^{t_U} D^m\phi_k(t)D^m\phi_l(t)dt = \begin{cases} 0, & k \neq l \\ \left(\frac{kw}{2}\right)^{2m}, & k = l; l, k = 1, \ldots, K, \end{cases}$

with

$$D^m\phi_0(t) = \begin{cases} 1/\sqrt{T}, & m = 0, \\ 0, & m > 0, \end{cases} \tag{4.18}$$

$$D^m\phi_k(t) = \frac{(k\omega/2)^m}{\sqrt{T/2}} \sin\left(k\omega t + \frac{m\pi}{2\omega}\right), k = 2r-1, r = 1, \ldots, K/2, \tag{4.19}$$

and

$$D^m\phi_l(t) = \frac{(l\omega/2)^m}{\sqrt{T/2}} \cos\left(l\omega t + \frac{m\pi}{2\omega}\right), l = 2r, r = 1, \ldots, K/2. \tag{4.20}$$

Proof.

1. (4.18) holds by direct verification so that it leaves to consider

$$\int_{t_L}^{t_U} \phi_0(t)\phi_0(t)dt = \int_{t_L}^{t_U} \frac{1}{T}dt = \frac{t_U - t_L}{T} = 1.$$

2. Using the relations for $a, b \in \mathbb{R}$

$$\int_0^{2\pi} \sin ax \sin bx dx = \int_0^{2\pi} \cos ax \cos bx dx = \begin{cases} 0, & a \neq b, \\ \pi, & a = b, \end{cases}$$

and $T = \frac{2\pi}{\omega}$ it follows that

$$\int_{t_L}^{t_U} \sin\left(k_1\omega t + \frac{m\pi}{2\omega}\right) \sin\left(k_2\omega t + \frac{m\pi}{2\omega}\right) dt = \begin{cases} 0, & k_1 \neq k_2, \\ T/2, & k_1 = k_2, \end{cases}$$

for $k_i = 2r_i - 1, r_i = 1, \ldots, K/2, i = 1, 2$, and

$$\int_{t_L}^{t_U} \cos\left(l_1\omega t + \frac{m\pi}{2\omega}\right) \cos\left(l_2\omega t + \frac{m\pi}{2\omega}\right) dt = \begin{cases} 0, & l_1 \neq l_2, \\ T/2, & l_1 = l_2, \end{cases}$$

for $l_i = 2r_i, r_i = 1, \ldots, K/2, i = 1, 2$, and by

$$\int_0^{2\pi} \sin ax \cos bx dx = 0$$

it follows that

$$\int_{t_L}^{t_U} \sin\left(k\omega t + \frac{m\pi}{2\omega}\right) \cos\left(l\omega t + \frac{m\pi}{2\omega}\right) dt = 0,$$

and the proposition gets directly verifiable.

□

For computing inner products of polynomial bases the next proposition can be used.

Proposition 5 *If $\phi_k(t) = (t-\omega)^k$ is a polynomial bases and $J = [a, b]$ then*

$$\begin{aligned} \int_J D^m\phi_k(t)D^m\phi_l(t)dt &= u\{(b-\omega)D^m\phi_k(b)D^m\phi_l(b)- \\ &\quad -(a-\omega)D^m\phi_k(a)D^m\phi_l(a)\} \end{aligned} \tag{4.21}$$

where

$$u = \frac{1}{k+l-2m+1}.$$

Proof.

$$\begin{aligned} &\int_J D^m\phi_k(t)D^m\phi_k(t)dt = \int_J D^m(t-\omega)^k D^m(t-\omega)^l dt \\ &= k(k-1)\dots(k-m+1)l(l-1)\dots(l-m+1)\int_{t_L}^{t_U}(t-\omega)^{k+l-2m}dt \\ &= \tfrac{k(k-1)\dots(k-m+1)l(l-1)\dots(l-m+1)}{k+l-2m+1}\left\{(t_U-\omega)^{k+l-2m+1} - (t_L-\omega)^{k+l-2m+1}\right\} \\ &= u\{(b-\omega)D^m\phi_k(b)D^m\phi_l(b) - (a-\omega)D^m\phi_k(a)D^m\phi_l(a)\} \end{aligned}$$

□

When computing inner products of B-spline bases one special analytic case appears if the order of derivative is equal to order of B-splines minus one. Then the derivatives are piecewise constant on each subinterval with positive support and their integrals are very easy to compute. It is easy to verify that for $J = [\tau_1, \tau_{n+K}]$

$$\int_J D^{k-1}B_{i,k}(t)D^{k-1}B_{j,k}(t)dt = \sum_{i=1}^{n+K-1}(\tau_{i+1}-\tau_i)D^{k-1}B_{i,k}(s)D^{k-1}B_{j,k}(s), \tag{4.22}$$

where $s \in [\tau_i, \tau_{i+1})$ can be chosen arbitrarily, since the derivatives are constant on these intervals. The product of two constant functions is again a constant function and multiplying it with the length of the interval gives the area among this product function. In the quantlet `inprod`, see Subsection 4.2.2, s is chosen to be the mean of the interval.

For all other B-spline bases one approach is that B-splines can be written in terms of piecewise polynomials. It holds for $\tau_l \leq t < \tau_{l+1}, l = 1, \ldots, n+K-1$ that

$$B_{i,k}(t) = \sum_{j=0}^{k-1} \frac{c_{ij}}{j!}(t-\tau_l)^j, \tag{4.23}$$

where $c_{ij} = D^j B_{i,k}(\tau_l)$. See de Boor (1978) for the proof. Furthermore, it is obvious that

$$D^m B_{i,k}(t) = \sum_{j=m}^{k-1} c_{ij}(t-\tau_l)^{j-m}, m < k, t \in [\tau_l, \tau_{l+1}). \tag{4.24}$$

The arising idea is to compute the integrals piecewise and to add them up afterwards. Here, the main advantage of the underlying quantlets concept becomes clear. The possibility to compute several derivatives simultaneously much simplifies computation time and effort in this procedure. For the calculation of inner products of B-spline bases the following proposition is needed.

Proposition 6 *For two B-splines and $m < k$ it holds*

$$\int_J D^m B_{i,k}(t) D^m B_{j,k}(t) dt = \sum_{l=1}^{n+K-1} \mathbf{1}_{(k-m)}^\top \mathbf{T}_{ijl} \mathbf{1}_{(k-m)}, \tag{4.25}$$

where the K dimensional quadratic matrix $\mathbf{T}_{ijl}$ has elements

$$\{\mathbf{T}_{ijl}\}_{rs} = c_{i,r+m-1} c_{j,s+m-1} \frac{(\tau_{l+1}-\tau_l)^{r+s-1}}{r+s-1}, r, s = 1, \ldots, k-m,$$

and $\mathbf{1}_{(k-m)} = (1, \ldots, 1)^\top$ is of dimension $k-m$.

Proof. Use (4.24) and that

$$\sum_{j=m}^{k-1} c_{ij}(t-\tau_l)^{j-m} = \sum_{j=1}^{k-m} c_{i,j+m-1}(t-\tau_l)^{j-1}, m < k, t \in [\tau_l, \tau_{l+1}).$$

Then the integral in (4.25) becomes

$$\sum_{l=1}^{n+K-1} \int_{\tau_l}^{\tau_{l+1}} \sum_{r=1}^{k-m} c_{i,r+m-1}(t-\tau_l)^{r-1} \sum_{s=1}^{k-m} c_{j,s+m-1}(t-\tau_l)^{s-1} dt. \tag{4.26}$$

The product of the sums in the integral in (4.26) can be written in matrix notation as

$$\sum_{r=1}^{k-m} c_{i,r+m-1}(t-\tau_l)^{r-1} \sum_{s=1}^{k-m} c_{j,s+m-1}(t-\tau_l)^{s-1} = \mathbf{1}_{(k-m)}^\top \mathbf{S}_{ijl} \mathbf{1}_{(k-m)},$$

with

$$\{\mathbf{S}_{ijl}\}_{rs} = c_{i,r+m-1}c_{j,s+m-1}(t-\tau_l)^{r+s-2}, r,s = 1,\ldots,k-m.$$

Since the integrand in (4.26) consists only on the sum of all elements of $\mathbf{S}_{ijl}$ just integrate the original function of each element as stored in $\mathbf{T}_{ijl}$

□

In all other cases a numerical solution is used provided by the Romberg algorithm and extended with a Neville extrapolation, following Press et al. (2002). The Romberg algorithm is a procedure for numerical integration. Gaussian Quadrature what is in fact superior to the Romberg algorithm need a transformation to the interval $[0,1]$ and also the computation of eigenvalues and eigenvectors, see Schwarz (1997). The latter can be very time-consuming when dealing with high dimensional matrices. Furthermore, since this will often be done in a numerical way an additional loss of precision is almost unavoidable. For ease of notation let

$$f(t) \stackrel{\text{def}}{=} \phi_i(t)\phi_j(t). \tag{4.27}$$

The starting point of the Romberg approach is the trapezoidal rule of numerical integration

$$\int_{t_L}^{t_U} f(t)dt \approx h_i[\frac{1}{2}f(t_L) + f(t_1) + \ldots + f(t_n) + \frac{1}{2}f(t_U)] \stackrel{\text{def}}{=} T(h_i), \tag{4.28}$$

where the integral of a function $f(t)$ is approximated by the sum of areas of the $n+1$ trapezoids, and h_i is the step-length, that means the distance between two adjoint evaluation points.

The Romberg method uses the results from m successive refinements of the trapezoidal rule. If at most $m+1$ iterations should be done, the step length of the mth iteration is

$$h_i = \frac{t_U - t_L}{2^i}, i = 0,1,2,\ldots,m. \tag{4.29}$$

The following relation shows that during the refinements only the new evaluation

points must be added

$$\begin{aligned} T(h_i) &= T\left(\frac{h_{i-1}}{2}\right) \\ &= \frac{1}{2}\left[T(h_{i-1}) + h_{i-1}\sum_{j=0}^{n-1} f\left(a + \frac{h_{i-1}}{2} + jh_{i-1}\right)\right], \end{aligned} \tag{4.30}$$

for $i = 1, 2, \ldots, m; n = 2^{i-1}$. The computational effort of the Romberg method is strongly lowered by using (4.30).

The successively lowering of the step-length h leads to apply the idea of Richardson's deferred approach to the limit: Perform the numerical integration for various values of the parameter h, and then extrapolate the result to the continuum limit $h = 0$. Here, Neville's algorithm can be used to extrapolate the successive refinements to zero step-size, see Press et al. (2002).

Neville's algorithm is a recursive way to use a given sequence of $N + 1$ points $\{x_i, y_i\}_{i=0}^{N}$, where x_i denotes the ith argument and y_i the ith function value, to estimate the function value $f(x)$ at some x, outside the range of the sequence. It is based on the relationship between a "daughter" P and its two "parents",

$$P_{i(i+1)\ldots,(i+m)} = \frac{(x - x_{i+m})P_{i(i+1)\ldots(i+m-1)} + (x_i - x)P_{(i+1)(i+2)\ldots(i+m)}}{x_i - x_{i+m}}, \tag{4.31}$$

where $P_i = y_i, i = 0, \ldots, N$, and $m = 1, \ldots, N$. Press et al. (2002) suppose an improvement on (4.31) to keep track of the small differences between parents and daughters as defined by

$$\begin{aligned} C_{m,i} &= P_{i\ldots(i+m)} - P_{i\ldots(i+m-1)}, \\ D_{m,i} &= P_{i\ldots(i+m)} - P_{(i+1)\ldots(i+m)}. \end{aligned} \tag{4.32}$$

Their values for $m = N$ are thus the error indication. Hence, with the help of Neville extrapolation we can decide, whether the numerical values of inner products have achieved a certain precision. For ease of use all these analytical and numerical methods are included in one quantlet called `inprod`.

4.2.2 The Quantlet inprod

```
inprodmat = inprod(fdo1,fdo2{,Lfd1{,Lfd2{,JMAX{,EPS}}}})
      computes inner products of functions by analytic or numerical
      integration, depending on the kind of input parameters
```

The quantlet `inprod` has the following input parameters

`fdo1` - The first fd or fdbasis object

`fdo2` - The second fd or fdbasis object

`Lfd1` - The first evaluation object. This can be a scalar, a $(r \times 1)$ vector of order of derivatives, a $(r \times 2)$ matrix in case of an LDO with constant coefficients, where the first column contains the coefficients, the second one the orders of derivatives. When to apply an LDO with variable coefficients Lfd1 must be an LDO object. The default value is Lfd1 = 2.

`Lfd2` - The second evaluation object. See `Lfd1`. The default value is Lfd2 = 2.

`JMAX` - Maximum number of iterations. The default value is JMAX = 15.

`EPS` - the fractional accuracy desired, as determined by the extrapolation error estimate. The default value is 10^{-4}.

The quantlet inprod returns an (nbasis1 $\times$ nbasis2 $\times$ x) array of inner products, where nbasis1, nbasis2 are the number of basis functions of fdo1 and fdo2, respectively, as given in their fdbasis.nbasis elements. x is equal to r, if Lfd1 and Lfd2 are vectors, otherwise x is equal to one. Note that if `Lfd1` is a $(r \times 1)$ vector then also `Lfd2` must be a $(r \times 1)$ vector.

As given above, the maximum number of iterations can be influenced by the user. `JMAX` should not be greater than 30 because the computation times increases exponentially. Alternatively, the accuracy `EPS` can be defined by the user. But as before, the increasing computation time should be kept in mind.

The quantlet computes inner product analytically if fdo1 = fdo2, Lfd1 = Lfd2, and `Lfd1` is no LDO object for all B-splines and all polynomials, and for Fourier bases if range = period. In all other cases the inner products are computed numerically using a version of the Romberg algorithm with combination of Neville extrapolation, see Subsection 4.2.1. Because of these special cases, the check whether `Lfd1` and `Lfd2` are matrices or LDO objects as in the `getbasismatrix` quantlet must be done again. If `fdo1` and/or `fdo2` are fd objects, the computation of their inner products is separated in two parts. The corresponding matrix **R** of inner products, see equation (4.17), will be computed as above, and afterwards multiplied with the coefficient vectors. In the scope of the hierarchical structure of the FDA quantlets (see Figure 2.2) this must actually be done by the `evalfd` quantlet, which will be introduced in Section 4.3. But the implemented version lowers computational effort, since less quantlets must be called. Hence, to use `evalfd` might be necessary, for example, if the coefficients are localized as in Section 4.4, or otherwise dynamically dependent on the argument.

To follow the spirit of simultaneous computation, `inprod` can also compute inner products for several derivatives simultaneously, what might be of less practical value.

If some analytic inner product solutions should be added this can be done easily by including the name of the basis in the if-condition before the special cases, when fdo1 = fdo2 and Lfd1 = Lfd2 and `Lfd1` is no LDO object should hold. Otherwise it can be included before the else-condition (above "all other cases: Romberg integration").

4.3 Creating fd objects in XploRe

```
fdobject = data2fd (y, argvals, fdbasis{, Lfd{, W{, lambda}}})
      converts an array y of function values plus an array argvals of
       argument values into a functional data object

evalmat = evalfd (evalarg, fd{, Lfd})
      evaluates an fd object, or the value of an LDO at argument
      values evalarg
```

An fd object can be created using the quantlet `data2fd`. This quantlet uses (4.8) to estimate the coefficients of the basis expansions. As a rule functions of t are observed only at discrete sampling values $\mathbf{t_i}, i = 1, \ldots, n$ as before, and these may or may not be equally spaced. But there may well be more than one function of t being observed. The syntax of the quantlet is

```
fdobject = data2fd (y, argvals, fdbasis{, Lfd{, W{, lambda}}})
```

with input parameters

`y` - (p1 × p2 × p3) array of (observed) functional values used to compute the functional data object. p1 is the number of observed values per replication, p2 is the number of replications, p3 is the number of variables.

`argvals` - $(n \times m)$ matrix of argument values, where $m = 1$ or $m =$ p2, respectively. If $m = 1$ the same arguments are applied to all replications and variables. Otherwise it must hold that $m =$ p2, then each replication is applied to a certain vector of arguments.

`basisfd` - an fdbasis object

`Lfd` - Identifies the penalty term. This can be a scalar (≥ 1) containing the order of derivative to penalize, or an LDO object if an LDO with variable coefficients is to be penalized. When applying an LDO with constant coefficients, Lfd is a $(r \times 2)$ matrix, where the first column contains the coefficients, the second one the orders of derivatives. The default value is Lfd = 2.

`W` - Weight matrix. The default value is the identity matrix.

`lambda` - Parameter for roughness penalty smoothing. If lambda is not specified it will be estimated by the data for each replication separately.

The quantlet `data2fd` returns an fd object, containing the (nbasis × p2 × p3) array of estimated coefficients and the underlying fdbasis object. `nbasis` is the number of basis functions as indicated in the fdbasis object. Note that `data2fd` requires the library `multi` because it uses the quantlet `spur`.

The input parameter `y` can be a vector, matrix or three-dimensional array, respectively. When some observations at some arguments are missing just insert `NaNs`. This is also the case when the number of observations vary between the replications. The algorithm filters out the `NaNs`. The input parameter `argvals` cannot contain `NaNs`, otherwise the `getbasismatrix` quantlet need to be readjusted in order to handle `NaNs`. If a functional value is missing for a certain argument, then just give an arbitrary value to the argument. The algorithm will filter out the pair of values in any case. It takes more computation time when `NaNs` are included in the functional data input because the compiler need to calculate the coefficients for each replication separately.

If lambda is not given then it will be computed by

$$\lambda = 10^{-4} \frac{\operatorname{tr}\left\{\Phi(\mathbf{t_i})^\top \Phi(\mathbf{t_i})\right\}}{\operatorname{tr}\mathbf{R}} \tag{4.33}$$

using the input parameters `y` and `argvals`. If necessary, lambda is computed individually for each replication. If lambda should be zero, then this must be given as input parameter. Note, if the number of basis functions is greater than the number of observations of a single replication, lambda will be estimated by (4.33) in any case, even if the corresponding input parameter is set equal to zero. Otherwise (4.8) could not be used because the outer product of the basis matrices in singular in this case.

The quantlet `evalfd` can be used to evaluate an fd object at a vector or matrix

of arguments. The syntax is

```
evalmat = evalfd (evalarg, fd{, Lfd})
```

with input parameters

`evalarg` - a $(p \times q)$ matrix of argument values

`fd` - an fd object

`Lfd` - This can be a scalar (≥ 1) containing the order of derivative to penalize, or an LDO object if an LDO with variable coefficients is to be penalized. When applying an LDO with constant coefficients, Lfd is a $(r \times 2)$ matrix, where the first column contains the coefficients, the second one the orders of derivatives. The default value is Lfd = 0.

This quantlet returns an $(p \times q \times x)$ array of functional values, where x is equal to r if Lfd is a vector and equal to one otherwise.

With the help of these two quantlets fd objects can now be created and evaluated. In the next two sections, some special smoothing methods will be introduced. These methods would need specific quantlets which are not implemented yet. Therefore, some remarks on the computational requirements are also made.

4.4 Localized Basis Function Estimators

The idea is to combine kernel estimators and basis function estimators to yield localized basis function estimators. A kernel estimator at a given point t is a linear combination of local observations,

$$\hat{X}_i(t) = \sum_{j=1}^{n} W_{hj}(t) y_{ij}, \tag{4.34}$$

where $W_{hj}(t)$ are suitably defined weight functions, such as

$$W_{hj}(t) = \frac{K_h(t - t_{ij})}{\sum_{r=1}^{n} K_h(t - t_{ir})}$$

for the Nadaraya-Watson estimator (see Nadaraya, 1964, Watson, 1964), and $K_h(\bullet) = \frac{1}{h}K(\frac{\bullet}{h})$ with some kernel function $K(\bullet)$ and the bandwidth h. Do not confuse the kernel function with the (representing) kernel introduced in Section 2.4. An overview on kernel density estimation and several kernel regression estimators is given in Härdle et al. (2004).

When using a localized basis function estimator the roughness penalty approach becomes

$$PENSSE_\lambda(X_i(t)|\mathbf{y_i}) = \{\mathbf{y_i} - \Phi(\mathbf{t_i})\mathbf{c_i}\}^\top \mathbf{W}(t)\{\mathbf{y_i} - \mathbf{\Phi}(\mathbf{t_i})\mathbf{c_i}\} + \lambda \mathbf{c}^\top \mathbf{R}\mathbf{c}, \quad (4.35)$$

where $\mathbf{W}(t) = \text{diag}\{W_{h1}(t), \ldots, W_{hn_i}(t)\}$. Hence, the vector of parameters which minimizes (4.35) is

$$\hat{\mathbf{c}}_\mathbf{i}(t) = \left\{\Phi(\mathbf{t_i})^\top \mathbf{W}(t)\Phi(\mathbf{t_i}) + \lambda \mathbf{R}\right\}^{-1} \Phi(\mathbf{t_i})^\top \mathbf{W}(t)\mathbf{y_i}, \quad (4.36)$$

and now depends on the argument t. The smoothing matrix becomes

$$\mathbf{S}_{\phi,\lambda,\mathbf{i}}(t) = \Phi(\mathbf{t_i}) \left\{\Phi(\mathbf{t_i})^\top \mathbf{W}(t)\Phi(\mathbf{t_i}) + \lambda \mathbf{R}\right\}^{-1} \Phi(\mathbf{t_i})^\top \mathbf{W}(t). \quad (4.37)$$

By the design of the underlying kernel estimators, the weight values $\mathbf{W}(t)$ have substantially positive support only for observations located close to the evaluation argument t at which the function is to be estimated. Consequently, $\hat{X}_i(t)$ is essentially a linear combination only of the observations $\mathbf{y_i}$ in the neighborhood of t.

One advantage of this approach is, that the basis can better approximate local features, since it only has to approximate data in a limited neighborhood of t. This can be done with a small number K of basis functions. The price to pay for this flexibility is that the expansion must essentially be carried out anew for each evaluation point t, see Ramsay & Silverman (1997).

The localized basis function estimator might not be well suited for further functional data analysis because of its high computational effort. That is why it has not been implemented yet in XploRe. When doing so the `data2fd` quantlet must be changed and also the `evalfd` quantlet, in order to be able to deal with variable

coefficients of the basis function expansions, furthermore all quantlets which need to evaluate the coefficient vector such as `inprod`.

4.5 The Regularized Basis Approach

In the roughness penalty approach the function $X_i(t)$ were forced to lie in a relatively low dimensional space, defined in terms of a suitable basis. In Section 4.4 we assume that not the whole function was in the span of a particular basis, but rather we considered a local basis expansion at any given point. In this section two complementary bases are used to fit the function to the observed data.

Assume that any function $X_i(t)$ of interest can be expanded in terms of two bases

$$X_i(t) = \sum_{j=1}^{J} d_{ij}\psi_j(t) + \sum_{k=1}^{K} c_{ik}\phi_k(t) \tag{4.38}$$

or in matrix terms as

$$X_i(t) = \Psi(t)\mathbf{d_i} + \Phi(t)\mathbf{c_i}, \tag{4.39}$$

with $\Psi(t) = \{\psi_1(t), \ldots, \psi_J(t)\}, \Phi(t) = \{\phi_1(t), \ldots, \phi_K(t)\}, \mathbf{d_i} = (d_{i1}, \ldots, d_{iJ})^\top$, and $\mathbf{c_i} = (c_{i1}, \ldots, c_{iK})^\top$. The basis functions $\Psi(t)$ are small in number and chosen to give reasonable account of the large-scale features of the data. For example, this could be polynomial bases of low degree. The complementary basis functions $\Phi(t)$ will generally be much larger in number, and are designed to catch local and other features not representable by the other basis functions $\Psi(t)$.

Now, assume that the roughness penalty must depend only on the coefficients of $\Phi(t)$ because $\Psi(t)$ are smooth enough. Otherwise, it would be sufficient to apply the usual roughness penalty approach to the combined basis of $\Phi(t)$ and $\Psi(t)$. Let $X_{iR}(t) = \Phi(t)\mathbf{c_i}$ and define a penalty term on $X_{iR}(t)$ using a linear differential operator

$$PEN_L(X_{iR}) = \int_J (LX_{iR}(t))^2 dt = \int_J \{\sum_{k=1}^{K} c_{ik} L\phi_k(t)\}^2 dt, \tag{4.40}$$

or expressed alternatively as

$$PEN_L(X_R) = \mathbf{c}^\top \mathbf{R}\mathbf{c}, \tag{4.41}$$

where the order K symmetric matrix $\mathbf{R}$ contains elements

$$\mathbf{R}_{kl} = \int_J L\phi_k(t)L\phi_l(t)dt = \langle L\phi_k, L\phi_l \rangle$$

as before. Then the composite roughness penalty criterion becomes

$$\begin{aligned} PENSSE_\lambda(X_i|\mathbf{y_i}) &= \{\mathbf{y_i} - \Psi_i(\mathbf{t_i})\mathbf{d_i} - \Phi_i(\mathbf{t_i})\mathbf{c_i}\}^\top \mathbf{W_i} \{\mathbf{y_i} - \Psi_i(\mathbf{t_i})\mathbf{d_i} - \Phi_i(\mathbf{t_i})\mathbf{c_i}\} \\ &+ \lambda \mathbf{c_i}^\top \mathbf{R} \mathbf{c_i}. \end{aligned} \tag{4.42}$$

Heckman and Ramsay (2000) show that (4.42) is furthermore a solution to the general penalized regression problem (4.7) without specifying any basis function expansion at all. Ramsay and Silverman (1997) show that by setting the partial derivatives of (4.42) equal to zero the solutions for $\mathbf{c_i}$ and $\mathbf{d_i}$ are

$$\hat{\mathbf{c}}_\mathbf{i} = \left\{ \Phi_i(\mathbf{t_i})^\top Q_{\Psi_i} \Phi_i(\mathbf{t_i}) + \lambda \mathbf{R} \right\}^{-1} \Phi_i(\mathbf{t_i})^\top Q_{\Psi_i} \mathbf{y_i} \tag{4.43}$$

and

$$\hat{\mathbf{d}}_\mathbf{i} = P_{\Psi_i}(\mathbf{t_i})(\mathbf{y_i} - \Phi_i(\mathbf{t_i})\mathbf{c_i}), \tag{4.44}$$

where $P_{\Psi_i} = \Psi_i(\mathbf{t_i})(\Psi_i(\mathbf{t_i})^\top \Psi_i(\mathbf{t_i}))^{-1} \Psi_i(\mathbf{t_i})^\top$ is the projection matrix, and $Q_{\Psi_i} = I - P_{\Psi_i}$ is the complementary projection. However, this resulting set of linear equations can be extremely ill-conditioned, and moreover, the matrix calculations require in general $O(n_i^3)$ calculations. Heckman & Ramsay (2000) provide an $O(n_i)$ algorithm.

Chapter 5

Summary and Outlook

The aim of this paper has been to introduce the concept of functional data, the basis function approach for their representation as smooth functions, and smoothing techniques for estimation out of discretely observed data. We showed the main properties of the most often used basis expansions, the basic principles of penalized regression and how to compute penalty matrices. The described system of quantlets for the interactive statistical computing environment XploRe provides the creation and computation of functional data (fd) objects and offers the starting point for further functional data analysis.

In a next step, methods used to display and summarize functional data are to be provided. Here the main advantage is that transformations such as variance, covariance, or mean functions can be applied directly to the `coef` element of the fd objects. As a result we do not need many specialized functions, except quantlets such as for computing correlation or covariance functions. Afterwards, quantlets for the typical FDA methods, as principal component analysis (PCA), principal differential analysis (PDA), or functional linear models, has to be developed.

Additionally, the already implemented smoothing methods can be extended. It can happen that a smoothing function is required to satisfy certain constraints. The most often constraints in the spirit of Ramsay (2003) are: (1) that the function be strictly positive, (2) that the function be strictly increasing or monotone, and (3) that the function be a probability density function. Just using the standard

smoothing functions above will often not work because there is no provision in them for forcing the functions to be constrained in any way. Ramsay (1998) proposes a methods under the constraint that the estimated smooth function is strictly monotone. Ramsay & Silverman (2002) describe an algorithm on how to estimate probability density functions using basis function expansions. Both methods are more computationally intensive because iterative methods of nonlinear regression need to be applied. As already mentioned, using wavelet bases for expansions might be another aspect for further extension of the existing system of quantlets.

The novel statistical technology of functional data analysis has already sparked a huge number of theoretical and applied essays. There are still a lot of perspectives for further research. The system of quantlets, presented in this thesis may support applications on functional data, as well as give impulses for future extensions.

Bibliography

Black, F. & Scholes, M. (1973). The pricing of options and corporate liabilities, *Journal of Political Economy* **81**: 637–654.

Cailliez, F. & Pagès, J.-P. (1976). *Introduction à l'Analyse des Données*, Paris: Société de Mathématiques Appliquées et de Sciences Humaines.

Craven, P. & Wahba, G. (1979). Smoothing noisy data with spline functions, *Numerische Mathematik* **31**: 377–403.

Curry, H. B. & Schoenberg, I. J. (1966). On Pólya frequency functions. IV: The fundamental spline functions and their limits, *Journal d'Analysée Mathématique* **17**: 71–107.

Daubechies, I. (1992). *Ten Lectures on Wavelets*, SIAM, Philadelphia.

Dauxois, J. & Pousse, A. (1976). Les analyses factorielles en calcul des probabilités et en statistique: Essai d'etude synthètique. Thèse d'état, l'Université Paul-Sabatier de Toulouse, France.

de Boor, C. (1978). *A Practical Guide to Splines*, Springer, New York.

Eubank, R. L. (1988). *Spline Smoothing and Nonparametric Regression*, Marcel Dekker, New York.

Green, P. J. & Silverman, B. W. (1994). *Nonparametric Regression and Generalized Linear Models. A roughness penalty approach*, Chapmann & Hall, London.

Härdle, W., Kerkyacharian, G., Picard, D. & Tsybakov, A. (1998). *Wavelets, Approximations, and Statistical Applications, Lecture Notes in Statistics* **129**, Springer Verlag, New York.

Härdle, W., Müller, M., Sperlich, S. & Werwatz, A. (2004). *Nonparametric and Semiparametric Models. An Introduction*, Springer, New York.

Hastie, T., Tibshirani, R. & Friedman, J. (2001). *The Elements of Statistical Learning. Data Mining, Inference, and Prediction*, Springer, New York.

Heckman, N. E. & Ramsay, J. O. (2000). Penalized regression with model-bases penalties, *The Canadian Journal of Statistics* **28**: 241–258.

Istrăţescu, V. I. (1987). *Inner Product Structures. Theory and Applications*, D. Reidel Publishing Company, Dordrecht.

Judge, G. G., Griffiths, W. E., Hill, R. C., Lütkepohl, H. & Lee, T.-C. (1988). *Introduction to the Theory and Practice of Econometrics*, John Wiley & Sons, New York.

Kimeldorf, G. S. & Wahba, G. (1970). A correspondence between Bayesian estimation on stochastic processes and smoothing by splines, *The Annals of Mathematical Statistics* **41**: 495–502.

Leurgans, S. E., Moyeed, R. A. & Silverman, B. W. (1993). Canonical correlation analysis when the data are curves, *Journal of the Royal Statistical Society, Series B* **55**: 725–740.

Nadaraya, E. A. (1964). On estimating regression, *Theory of Probability and its Applications* **10**: 186–190.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. & Flannery, B. P. (2002). *Numerical Recipes in C++,* 2nd edition, Cambridge University Press, Cambridge.

Ramsay, J. O. (1982). When the data ara functions, *Psychometrika* **47**: 379–396.

Ramsay, J. O. (1997). Functional data analysis. In S. Kotz, C. B. Read & D. L. Banks (Eds.), *Encyclopedia of Statistical Sciences*, Wiley, New York.

Ramsay, J. O. (1998). Estimating smooth monotone functions, *Journal of the Royal Statistical Society, Series B* **60**: 365–375.

Ramsay, J. O. (2003). *Matlab, R and S-PLUS Functions for Functional Data Analysis*, available at ftp://ego.psych.mcgill.ca/pub/ramsay/FDAfuns.

Ramsay, J. O. & Dalzell, C. J. (1991). Some tools for functional data analysis, *Journal of the Royal Statistical Society, Series B* **53**: 539–572.

Ramsay, J. O. & Silverman, B. W. (1997). *Functional Data Analysis*, Springer, New York.

Ramsay, J. O. & Silverman, B. W. (2002). *Applied Functional Data Analysis. Methods and Case Studies*, Springer, New York.

Reinsch, C. (1967). Smoothing by spline functions. *Numerische Mathematik* **10**: 177–183.

Rice, J. A. & Silverman, B. W. (1991). Estimating the mean and covariance structure nonparametrically when the data are curves, *Journal of the Royal Statistical Society, Series B* **53**: 233–244.

Schumaker, L. L. (1981). *Spline Functions: Basic Theory*, John Wiley & Sons, New York.

Schwarz, H. R. (1997). *Numerische Mathematik*, B. G. Teubner, Stuttgart.

Silverman, B. W. (1985). Some aspects of the spline smoothing approach to non-parametric regression curve fitting, *Journal of the Royal Statistical Society, Series B* **47**: 1–52.

Simmons, G. F. (1963). *Introduction to Topology and Modern Analysis*, McGraw-Hill Kogakusha, Tokyo.

Wahba, G. (1990). *Spline models for observational data*, Society for Industrial and Applied Mathematics, Philadelphia.

Watson, G. S. (1964). Smooth regression analysis, *Sankhyā, Series A* **26**: 359–372.

Electronic Source

The included CD-ROM contains the quantlets described in this thesis, as well as all examples, and tex-files sources.

- "FDA Quantlets" contains all FDA quantlets described in this thesis:

  `Bsplineevalgd.xpl`

  `Fourierevalgd.xpl`

  `polyevalgd.xpl`

  `createfdbasis.xpl`

  `getbasismatrix.xpl`

  `data2fd.xpl`

  `inprod.xpl`

  `evalfd.xpl`

  `createldo.xpl`

- "Examples" contains all examples and quantlets used for figures:

  `LDOexample.xpl`

  `fb1.xpl`

  `pb1.xpl`

  `pcp1.xpl`

  `bspl1.xpl`

  `bspl2.xpl`

`bspl3.xpl`

`bspl4.xpl`

`bspl5.xpl`

`bsplderiv1.xpl`

`bsplderiv2.xpl`

`tensor1.xpl`

`tensor2.xpl`

`rp1.xpl`

- "C++" contains all files of the dynamically linked library
- "TeX-files" contains all file of the thesis produced by LaTeX. Note, the pdf-file is NOT the web-supported version!!!